



Adapters Guide

Sybase Event Stream Processor

5.0

DOCUMENT ID: DC01615-01-0500-05

LAST REVISED: June 2012

Copyright © 2012 by Sybase, Inc. All rights reserved.

This publication pertains to Sybase software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

Upgrades are provided only at regularly scheduled software release dates. No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of Sybase, Inc.

Sybase trademarks can be viewed at the Sybase trademarks page at <http://www.sybase.com/detail?id=1011207>. Sybase and the marks listed are trademarks of Sybase, Inc. ® indicates registration in the United States of America.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world.

Java and all Java-based marks are trademarks or registered trademarks of Oracle and/or its affiliates in the U.S. and other countries.

Unicode and the Unicode Logo are registered trademarks of Unicode, Inc.

All other company and product names mentioned may be trademarks of the respective companies with which they are associated.

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

CHAPTER 1: Introduction	1
Input and Output Adapters	1
Internal and External Adapters	2
Custom Adapters	3
Working with Adapters	3
Publishing Data with Output Adapters	4
Datatypes	5
Adapter Parameters Datatypes	8
Date and Timestamp Formats for Input Adapters	11
Date and Timestamp Formats for Output Adapters	12
CHAPTER 2: Adapters Supported by Event Stream Processor	15
Adapter Summary	15
Editing Adapter Property Sets	17
Internal Adapters	18
AtomReader Input Adapter	18
Database Adapter	20
Database Input Adapter	20
Database Output Adapter	22
Datatype Mapping for the Database Adapter	26
File CSV Input Adapter	32
File CSV Output Adapter	35
File XML Input Adapter	37
File XML Output Adapter	41
File FIX Input Adapter	42
Datatype Mapping for the File FIX Input Adapter	43
File FIX Output Adapter	44

Datatype Mapping for the File FIX Output Adapter	45
JMS Adapter	46
Configuring a Queuing System for JMS Adapter	46
JMS CSV Input Adapter	47
JMS CSV Output Adapter	50
JMS Custom Input Adapter	55
JMS Custom Output Adapter	59
JMS FIX Input Adapter	64
JMS FIX Output Adapter	66
JMS Object Array Input Adapter	70
JMS Object Array Output Adapter	74
JMS XML Input Adapter	79
JMS XML Output Adapter	82
Random Tuples Generator Input Adapter	87
Socket FIX Input Adapter	92
Datatype Mapping for the Socket FIX Input Adapter	94
Socket FIX Output Adapter	94
Datatype Mapping for the Socket FIX Output Adapter	96
Socket (As Client) CSV Input Adapter	96
Socket (as Client) CSV Output Adapter	99
Socket (As Client) XML Input Adapter	101
Socket (As Client) XML Output Adapter	103
Socket (As Server) XML Input Adapter	104
Socket (As Server) XML Output Adapter	106
Socket (As Server) CSV Input Adapter	107
Socket (As Server) CSV Output Adapter	109
SMTP Output Adapter	110
Sybase IQ Output Adapter	115
Datatype Mapping for the Sybase IQ Adapter ..	120
WebSphere MQ Adapter	120
WebSphere MQ Input Adapter	121

WebSphere MQ Output Adapter	123
Queue Configuration	126
External Adapters	127
ESP Add-In for Microsoft Excel	127
Connection Wizard	127
Subscription Wizard	129
Publication Wizard	130
Automatic Publishing	132
Subscription Queries	134
Applying a Query	134
Known Issues and Limitations	135
FIX Adapter	135
Supported FIX Versions	136
Control Flow	136
Data Streams	138
Adapters and Sessions	141
Message Flow	142
Datatype Mapping for the FIX Adapter	143
Setting the JAVA_HOME Environment Variable	143
Configuration	143
Operation	164
Examples	166
Flex Adapter	175
Control Flow	175
Message Flow	176
Stream Handler	177
Setting the JAVA_HOME Environment Variable	179
Configuration	179
Operation	184
Example: Sending a Subscription Request	186
HTTP Output Adapter	188
Control Flow	188
Message Flow	190
Setting the JAVA_HOME Environment Variable	191
Configuration	191

Operation	198
Example: Sending, Receiving, and Viewing	
Data	200
KDB Adapter	202
Control Flow	202
Datatype Mapping for the KDB Adapter	202
KDB Input Adapter	204
KDB Output Adapter	209
Log File Input Adapter	214
Configuration	215
Properties	216
Starting the Adapter from the Command Line ..	218
NYSE Technologies Input Adapter	219
Control Flow	219
Watchlists	221
Data Streams	224
Stale Records	226
Message Flow	227
Datatype Mapping for the NYSE Adapter	228
Setting the JAVA_HOME Environment Variable	
.....	228
Configuration	228
Operation	237
Example: Subscribing to and Publishing Data ..	240
Open Adapter	241
Datatype Mapping for the Open Adapter	242
Setting the JAVA_HOME Environment Variable	
.....	243
Configuration	243
Starting the Open Adapter	284
Monitoring the Open Adapter	284
Examples	289
RAP Adapter	303
Start Command	303
Stop Command	304

Datatype Mapping for the RAP Adapter	304
Configuration	306
Operation	318
Replication Server Adapter	320
Configuring the Adapter on the Replication Server Workstation	320
Configuring the Adapter on an Event Streaming Processor Workstation	322
Defining a Persistent rs_lastcommit	326
Supported Datatypes	327
Performance Tips	327
Reuters Marketfeed Adapter	327
Requirements	328
General Configuration	328
Input Adapter Configuration	331
Output Adapter Configuration	342
Creating a Subordinate Map File	348
Performance Tuning	350
Command Usage	352
Environment Variables	354
Input Adapter Map File XML Syntax	355
Output Adapter Map File XML Syntax	382
Logging Facilities	395
Reuters OMM Adapter	401
Requirements	401
General Configuration	402
Input Adapter Configuration	405
Output Adapter Configuration	415
Split Adapter Map Files	419
Command Usage	420
Environment Variables	424
Input Adapter Map File	426
Output Adapter Map File XML Syntax	458
Logging Facilities	469
RTView Adapter	475

Datatype Mapping for the RTView Adapter	475
Installing the RTView Adapter	476
Configuration: Creating and Updating a Sybase Connection	476
Operation	479
Running the Publisher Example	485
Running the Subscriber Example	486
Known Limitations	487
TIBCO Rendezvous Adapter	488
Control Flow	488
Data Streams	490
Message Flow	490
Datatype Mapping for the TIBCO Rendezvous Adapter	491
Setting the JAVA_HOME Environment Variable	492
Configuration	492
Operation	501
Example: Subscribing and Publishing	503
CHAPTER 3: Custom Adapters	505
Custom Internal Adapters	505
The Adapter Shared Utility Library	505
Callback Functionality	506
Sample Model File	506
The Adapter Configuration File	506
Adapter Life Cycle Functions	507
Adapter Setup Functions	508
Miscellaneous Functions	508
Adapter Run States	509
Schema Discovery for Internal Custom Adapters	510
Sample Custom Internal Adapter Implementation	510
Custom External Adapters	517
External Adapter Configuration File	518

External Adapter Properties	521
External Adapter Commands	521
User-Defined Parameters and Parameter Substitution	523
Auto-Generated Parameter Files	525
configFilename Parameter	526
Custom External Parameter Datatypes	526
Creating Custom External Adapters	527
Java External Adapters	527
C/C++ External Adapters	531
.Net External Adapters	534
 CHAPTER 4: Schema Discovery	539
Adapters that Support Schema Discovery	539
 CHAPTER 5: Guaranteed Delivery	543
Log Window	544
Truncate Window	545
 Index	547

Contents

CHAPTER 1 Introduction

Sybase® Event Stream Processor supplies an extensive set of input and output adapters that you can use to subscribe to and publish data, including external process adapters. Event Stream Processor also provides several SDKs that enable you to write an adapter.

The supplied adapters support numerous datatypes, as well as providing datatype mapping for unsupported types.

Input and Output Adapters

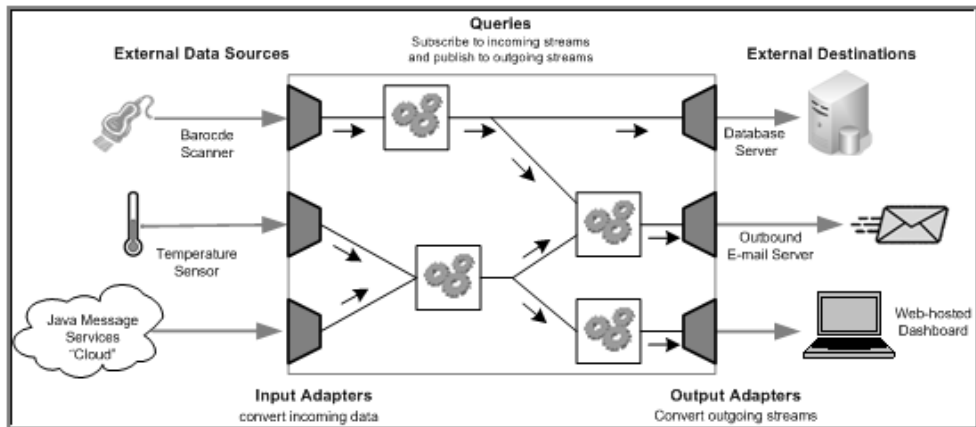
Input and output adapters enable Event Stream Processor to send and receive messages from dynamic and static external sources and destinations.

External sources or destinations can include:

- Data feeds
- Sensor devices
- Messaging systems
- Radio frequency identification (RFID) readers
- E-mail servers
- Relational databases

Input adapters connect to an external datasource and translate incoming messages from the external sources into a format that is accepted by the Event Stream Processor Server. Output adapters translate rows published by Event Stream Processor 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 compatible with Event Stream Processor. After the data is processed using various queries within Event Stream Processor, output adapters convert the result rows into updates that are sent to an external database server, e-mail server, and Web services dashboard.

Figure 1: Adapters in Event Stream Processor

Internal and External Adapters

An adapter that runs as part of the Server is called an internal adapter. An adapter that runs as a separate process is called an external adapter.

Internal adapters generally run faster because the Server can get data from (or to) the adapter with less overhead. Internal adapters are started by the Server when it starts the corresponding project (query module). The adapter is recognized by Sybase Event Stream Processor Studio, and from inside Studio, you can attach the adapter to a stream by selecting the adapter from a menu of adapters. A disadvantage of internal adapters is that if the adapter crashes, it may crash the Server as well.

External adapters have more flexibility than internal adapters and can run on a different machine than the Server. Like internal adapters, external adapter can be configured using Studio, and started and stopped by the Server as long as they are installed with an adapter configuration file.

External adapters can be either "managed" or "unmanaged". Managed external adapters provide an adapter configuration file (.cnxml) that can be configured using Studio, referenced in a CCL **ATTACH ADAPTER** statement, and can be started and stopped by the Server, behaving very similarly to an internal adapter. Unmanaged external adapters are not referenced in a CCL **ATTACH ADAPTER** statement, and are not managed by the Server. You start, stop, and configure these adapters independently.

Custom Adapters

In addition to the adapters provided by Event Stream Processor, you can write your own adapters to integrate into the Server. You can design adapters to handle a variety of external requirements that the standard adapters cannot manage.

Event Stream Processor provides a variety of SDKs that allow you to write adapters in a number of programming languages, including:

- C
- C++
- Java
- .NET (C#, Visual Basic, and so on)

For versions supported for SDKs, see the *Installation Guide*.

See also

- *Custom External Adapters* on page 517
- *Custom Internal Adapters* on page 505
- *Java External Adapters* on page 527
- *C/C++ External Adapters* on page 531
- *.Net External Adapters* on page 534
- *Creating Custom External Adapters* on page 527

Working with Adapters

Pre-planning and configuration steps to consider when using Event Stream Processor-supplied and custom adapters.

Overview of typical tasks to perform before attaching an input adapter to the Server, as well as an introduction to the CCL **ATTACH ADAPTER** statement.

Note: Unmanaged adapters are not referenced in the CCL **ATTACH ADAPTER** statement, and need to be configured, started, and stopped independently.

Detailed information on configuring individual adapters, datatype mapping, and schema discovery are contained in this guide. You can also refer to the CCL queries, **ATTACH ADAPTER** statement, **CREATE SCHEMA** statement, and **Parameters** topics in the *CCL Programmers Guide*.

1. Assess the input data. Determine which sets or subsets of data you want to pull into the Server

2. Choose an input adapter for your task.

If the datasource uses datatypes that are not supported by the Server, the Server maps the data to an accepted datatype. Review the associated mapping description for your adapter in this guide.

3. Configure your adapter as required.

4. Create a stream or window and use the **CREATE SCHEMA** statement to define the structure for incoming data.

5. Use the **ATTACH ADAPTER** statement to attach your adapter to the Server stream or window, and set values for the adapter properties.

To declare default parameters for your adapter properties, use the **DECLARE** block and **parameters** 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 only bind declared parameters to a new value when a module or project is loaded.

6. Run your data queries and perform analysis.

7. Publish your results.

Publishing Data with Output Adapters

An overview of typical tasks to perform before attaching an output adapter to an external data destination.

1. Assess the output data. Determine which sets or subsets of data you want to send to an external data destination.

2. Choose an output adapter for your task.

If the output destination uses datatypes that are not supported by the Server, the Server maps the data to an accepted datatype. Review the associated mapping description for your adapter in this guide to ensure that the resulting datatype is permitted by the external data destination.

3. Configure the adapter as required.

4. Create an output stream or window and use the **CREATE SCHEMA** statement to define the structure for outgoing data.

5. 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 **parameters** 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 only bind declared parameters to a new value when a module or project is loaded.

6. Publish your results.

Datatypes

Sybase Event Stream Processor supports integer, float, string, money, long, and timestamp datatypes for all of its components.

Datatype	Description
integer	<p>A signed 32-bit integer. 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.</p> <p>To initialize a variable, parameter, or column with a value of -2147483648, specify $(-2147483647) - 1$ to avoid CCL compiler errors.</p>
long	<p>A signed 64-bit integer. The range of allowed values is -9223372036854775808 to +9223372036854775807 (-2^{63} to 2^{63-1}).</p> <p>To initialize a variable, parameter, or column with a value of -9223372036854775808, specify $(-9223372036854775807) - 1$ to avoid CCL compiler errors.</p>
float	<p>A 64-bit numeric floating point with double precision. The range of allowed values is approximately -10^{308} through $+10^{308}$.</p>
string	<p>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.</p>
money	<p>A legacy datatype maintained for backward compatibility. It is a signed 64-bit integer that supports 4 digits after the decimal point. Currency symbols and commas are not supported in the input data stream.</p>

Datatype	Description
money (n)	<p>A signed 64-bit numerical value that supports varying scale, from 1 to 15 digits after the decimal point. Currency symbols and commas are not supported in the input data stream, however, decimal points are.</p> <p>The supported range of values change, depending on the specified scale.</p> <p>money (1): -922337203685477580.8 to 922337203685477580.7</p> <p>money (2): -92233720368547758.08 to 92233720368547758.07</p> <p>money (3): -9223372036854775.808 to 9223372036854775.807</p> <p>money (4): -922337203685477.5808 to 922337203685477.5807</p> <p>money (5): -92233720368547.75808 to 92233720368547.75807</p> <p>money (6): -9223372036854.75808 to 9223372036854.75807</p> <p>money (7): -922337203685.4775808 to 922337203685.4775807</p> <p>money (8): -92233720368.54775808 to 92233720368.54775807</p> <p>money (9): -9223372036.854775808 to 9223372036.854775807</p> <p>money (10): -922337203.6854775808 to 922337203.6854775807</p> <p>money (11): -92233720.36854775808 to 92233720.36854775807</p> <p>money (12): -9223372.036854775808 to 9223,372.036854775807</p> <p>money (13): -922337.2036854775808 to 922337.2036854775807</p> <p>money (14): -92233.72036854775808 to 92233.72036854775807</p> <p>money (15): -9223.372036854775808 to 9223.372036854775807</p> <p>To initialize a variable, parameter, or column with a value of -92,233.72036854775807, specify (-9...7) -1 to avoid CCL compiler errors.</p> <p>Specify explicit scale for money constants with Dn syntax, where n represents the scale. For example, 100.1234567D7, 100.12345D5.</p> <p>Implicit conversion between money (n) types is not supported because there is a risk of losing range or scale. Perform the cast function to work with money types that have different scale.</p>

Datatype	Description
bigdatetime	<p>Timestamp with microsecond precision. The default format is YYYY-MM-DDTHH:MM:SS:SSSSS.</p> <p>All numeric datatypes are implicitly cast to <code>bigdatetime</code>.</p> <p>The rules for conversion vary for some datatypes:</p> <ul style="list-style-type: none"> • All <code>boolean</code>, <code>integer</code>, and <code>long</code> values are converted in their original format to <code>bigdatetime</code> • Only the whole-number portions of <code>money(n)</code> and <code>float</code> values are converted to <code>bigdatetime</code>. Use the cast function to convert <code>money(n)</code> and <code>float</code> values to <code>bigdatetime</code> with precision. • All <code>date</code> values are multiplied by 1000000 and converted to microseconds to satisfy <code>bigdatetime</code> format. • All <code>timestamp</code> values are multiplied by 1000 and converted to microseconds to satisfy <code>bigdatetime</code> format.
timestamp	Timestamp with millisecond precision. The default format is YYYY-MM-DDTHH:MM:SS:SSS.
date	Date with second precision. The default format is YYYY-MM-DDTHH:MM:SS.

Datatype	Description
interval	<p>A signed 64-bit integer that represents the number of microseconds between two timestamps. Specify an <code>interval</code> using multiple units in space-separated format, for example, "5 Days 3 hours 15 Minutes". External data that is sent to an interval column is assumed to be in microseconds. Unit specification is not supported for <code>interval</code> values converted to or from <code>string</code> data.</p> <p>When an <code>interval</code> is specified, the given interval must fit in a 64-bit integer (<code>long</code>) when it is converted to the appropriate number of microseconds. For each <code>interval</code> unit, the maximum allowed values that fit in a long when converted to microseconds are:</p> <ul style="list-style-type: none"> • MICROSECONDS (MICROSECOND, MICROS): +/- 9223372036854775807 • MILLISECONDS (MILLISECOND, MILLIS): +/- 9223372036854775 • SECONDS(SECOND, SEC): +/- 9223372036854 • MINUTES(MINUTE, MIN): +/- 153722867280 • HOURS(HOUR,HR): +/- 2562047788 • DAYS(DAY): +/- 106751991 <p>The values in parentheses are alternate names for an <code>interval</code> unit. When the maximum value for a unit is specified, no other unit can be specified or it causes an overflow. Each unit can be specified only once.</p>
binary	Represents a raw binary buffer. Maximum length of value is platform-dependent, but can be no more than 65535 bytes. NULL characters are permitted.
boolean	Value is true or false. The format for values outside of the allowed range for <code>boolean</code> is 0/1/false/true/y/n/on/off/yes/no, which is case-insensitive.

Adapter Parameters Datatypes

A comprehensive list of datatypes you can use with adapters supplied by Event Stream Processor, or any custom internal or external adapters you create.

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

Note: This table includes all the adapter related datatypes supported by Event Stream Processor. For more information on specific datatypes supported by an adapter, as well as its datatype mapping description, see the section on that adapter.

Datatype	Description
boolean	Value is true or false. The format for values outside of the allowed range for <code>boolean</code> is <code>0/1/false/true/y/n/on/off/yes/no</code> , which is case insensitive.
choice	A list of custom values from which a user would select one value.
configFilename	Variable-length character string, with byte values encoded in UTF-8. Maximum string length is platform-dependent, but no more than 65535 bytes.
directory	Variable-length character string, with byte values encoded in UTF-8. Maximum string length is platform-dependent, but no more than 65535 bytes.
double	Floating point value. The range of allowed values is $2.22507e-308$ to $1.79769e+308$
filename	Variable-length character string, with byte values encoded in UTF-8. Maximum string length is platform-dependent, but no more than 65535 bytes.
int	<p>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.</p> <p>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.</p>
password	<p>Variable-length character string, with byte values encoded in UTF-8. Maximum string length is platform-dependent, but no more than 65535 bytes.</p> <hr/> <p>Note: While entering value for this field, user can see '*' for every character.</p> <hr/>
permutation	<p>Variable-length character string, with byte values encoded in UTF-8. Maximum string length is platform-dependent, but no more than 65535 bytes.</p> <hr/> <p>Note: This datatype is not supported for custom external adapters.</p> <hr/>

Datatype	Description
range	An integer value for which user can define lower and upper limits. e.g. <pre><Parameter id="port" label="KDB Port" descr="IP port of the database listener" type="range" rangeLow="0" rangeHigh="65535" default="5001" use="required" /></pre>
query	A string value Studio creates from the tablename. <hr/> Note: This datatype is not supported for custom external adapters.
runtimeDirectory	Variable-length character string, with byte values encoded in UTF-8. Maximum string length is platform-dependent, but no less than 65535 bytes. <hr/> Note: This datatype is not supported for custom external adapters.
runtimeFilename	Runtime filename, if different from discovery time filename. Variable-length character string, with byte values encoded in UTF-8. Maximum string length is platform-dependent, but no more than 65535 bytes. <hr/> Note: This datatype is not supported for custom external adapters.
string	Variable-length character string, with byte values encoded in UTF-8. Maximum string length is platform-dependent, but no more than 65535 bytes.
tables	This is list of choices returned by getTables() defined in adapter.
text	A value capable of storing multiline text. <hr/> Note: This datatype is not supported for custom external adapters.
uint	Positive integer value. The range of allowed values is 0 to 0xffffffff.

See also

- *Custom External Parameter Datatypes* on page 526
- *Internal Adapters* on page 18
- *External Adapters* on page 127
- *Chapter 3, Custom Adapters* on page 505

Date and Timestamp Formats for Input Adapters

Sybase supports numerous formats for date and timestamp datatypes.

Use the info below to create a custom format for your date and timestamp datatypes.

Character	Description
%a	The day of the week, using the locale's weekday names. You can specify either the abbreviated or full name.
%A	Equivalent to %a.
%b	The month, using the locale's month names. You can specify either the abbreviated or full name.
%B	Equivalent to %b.
%c	The locale's appropriate date and time representation.
%C	The century number [00,99]. Leading zeros are permitted but not required.
%d	The day of the month [01,31]. Leading zeros are permitted but not required.
%D	The date as %m / %d / %y.
%e	Equivalent to %d.
%h	Equivalent to %b.
%H	The hour (24-hour clock) [00,23]. Leading zeros are permitted but not required.
%I	The hour (12-hour clock) [01,12]. Leading zeros are permitted but not required.
%j	The day number of the year [001,366]. Leading zeros are permitted but not required.
%m	The month number [01,12]. Leading zeros are permitted but not required.
%M	The minute [00,59]. Leading zeros are permitted but not required.
%n	Any white space.
%p	The locale's equivalent of a.m or p.m.
%r	12-hour clock time using the AM/PM notation.
%R	The time as %H : %M.
%S	The seconds [00,60]. Leading zeros are permitted but not required.

Character	Description
%t	Any white space.
%T	The time as %H : %M : %S.
%U	The week number of the year (Sunday as the first day of the week) as a decimal number [00,53]. Leading zeros are permitted but not required.
%w	The weekday as a decimal number [0,6], with 0 representing Sunday. Leading zeros are permitted but not required.
%W	The week number of the year (Monday as the first day of the week) as a decimal number [00,53]. Leading zeros are permitted but not required.
%x	The date, using the locale's date format.
%X	The time, using the locale's time format.
%y	The year within the century. When a century is not otherwise specified, values in the range [69,99] shall refer to years 1969 to 1999 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive. Leading zeros are permitted but not required.
%Y	The year, including the century. For example, 1988.
%%	Replaced by %.

Date and Timestamp Formats for Output Adapters

Sybase supports numerous formats for date and timestamp datatypes.

Use the info below to create a custom format for your date and timestamp datatypes.

Character	Description
%a	The locale's abbreviated weekday name.
%A	The locale's full weekday name.
%b	The locale's abbreviated month name.
%B	The locale's full month name.
%c	The locale's appropriate date and time representation.
%C	The year divided by 100, and truncated to an integer as a decimal number [00,99].
%d	The day of the month as a decimal number [01,31].
%D	Equivalent to %m / %d / %y.

Character	Description
%e	The day of the month as a decimal number [1,31]. A single digit is preceded by a space.
%F	Equivalent to %Y - %m - %d. This is the ISO 8601:2000 standard date format.
%g	The last 2 digits of the week-based year, as a decimal number [00,99].
%G	The week-based year as a decimal number. For example, 1977.
%h	Equivalent to %b.
%H	The hour (24-hour clock) as a decimal number [00,23].
%I	The hour (12-hour clock) as a decimal number [01,12].
%j	The day of the year as a decimal number [001,366].
%m	The month as a decimal number [01,12].
%M	The minute as a decimal number [00,59].
%n	A <newline>.
%p	The locale's equivalent of either a.m. or p.m.
%r	The time in a.m. and p.m. notation.
%R	The time in 24-hour notation (%H : %M).
%S	The second as a decimal number [00,60].
%t	A <tab>.
%T	The time in the following format %H : %M : %S.
%u	The weekday as a decimal number [1,7], with 1 representing Monday.
%U	The week number of the year as a decimal number [00,53]. The first Sunday of January is the first day of week 1, and days in the new year before this are in week 0.
%V	The week number of the year (Monday as the first day of the week) as a decimal number [01,53]. If the week containing 1 January has four or more days in the new year, then it is considered week 1. Otherwise, it is the last week of the previous year, and the next week is week 1. Both January 4th and the first Thursday of January are always in week 1.
%w	The weekday as a decimal number [0,6], with 0 representing Sunday.

CHAPTER 1: Introduction

Character	Description
%W	The week number of the year as a decimal number [00,53]. The first Monday of January is the first day of week 1, and days in the new year before this are in week 0.
%x	The locale's appropriate date representation.
%X	The locale's appropriate time representation.
%y	The last two digits of the year as a decimal number [00,99].
%Y	The year as a decimal number. For example, 1997.
%z	The offset from UTC in the ISO 8601:2000 standard format (+hhmm or -hhmm), or by no characters if no time zone is determinable. For example, "-0430" means 4 hours 30 minutes behind UTC (west of Greenwich).
%Z	The time zone name or abbreviation, or by no bytes if no time zone information exists.
%%	Replaced by %.

Adapters Supported by Event Stream Processor

Event Stream Processor supports various internal and external adapters.

Unless otherwise noted, these adapters support the same platforms and operating systems as the Server and Studio. For information, see the *Event Stream Processor Installation Guide*.

Adapter Summary

Summary on adapters supported by Event Stream Processor. Includes the adapter type, and whether the adapter is managed, Studio configurable, and supports guaranteed delivery.

Adapter	Type	Managed	Studio Configurable	Supports Guaranteed Delivery
AtomReader Input	Internal	Yes	Yes	No
Database Input	Internal	Yes	Yes	No
Database Output	Internal	Yes	Yes	No
File CSV Input	Internal	Yes	Yes	No
File CSV Output	Internal	Yes	Yes	No
File XML Input	Internal	Yes	Yes	No
File XML Output	Internal	Yes	Yes	No
File FIX Input	Internal	Yes	Yes	No
File FIX Output	Internal	Yes	Yes	No
JMS CSV Input	Internal	Yes	Yes	Yes
JMS CSV Output	Internal	Yes	Yes	Yes
JMS Custom Input	Internal	Yes	Yes	Yes
JMS Custom Output	Internal	Yes	Yes	Yes
JMS FIX Input	Internal	Yes	Yes	Yes
JMS FIX Output	Internal	Yes	Yes	Yes

CHAPTER 2: Adapters Supported by Event Stream Processor

Adapter	Type	Managed	Studio Configurable	Supports Guaranteed Delivery
JMS Object Array Input	Internal	Yes	Yes	Yes
JMS Object Array Output	Internal	Yes	Yes	Yes
JMS XML Input	Internal	Yes	Yes	Yes
JMS XML Output	Internal	Yes	Yes	Yes
Random Tuples Generator Input	Internal	Yes	Yes	No
Socket FIX Input	Internal	Yes	Yes	No
Socket FIX Output	Internal	Yes	Yes	No
Socket (as Client) CSV Input	Internal	Yes	Yes	No
Socket (as Client) CSV Output	Internal	Yes	Yes	No
Socket (as Client) XML Input	Internal	Yes	Yes	No
Socket (as Client) XML Output	Internal	Yes	Yes	No
Socket (as Server) XML Input	Internal	Yes	Yes	No
Socket (as Server) XML Output	Internal	Yes	Yes	No
Socket (as Server) CSV Input	Internal	Yes	Yes	No
Socket (as Server) CSV Output	Internal	Yes	Yes	No
SMTP Output	Internal	Yes	Yes	No
Sybase IQ Output	Internal	Yes	Yes	No
WebSphere MQ Input	Internal	Yes	Yes	Yes
WebSphere MQ Output	Internal	Yes	Yes	Yes
FIX Input	External	Yes	Yes	No

Adapter	Type	Managed	Studio Configurable	Supports Guaranteed Delivery
FIX Output	External	Yes	Yes	No
Flex Output	External	No	No	No
HTTP Output	External	Yes	Yes	No
KDB Input	External	Yes	Yes	No
KDB Output	External	Yes	Yes	No
Log File Input	External	No	No	No
NYSE Technologies Input	External	Yes	Yes	No
Open Input	External	No	No	No
Open Output	External	No	No	No
RAP Output	External	No	No	No
Replication Server Input	External	Yes	Yes	Yes
Reuters Marketfeed Input	External	No	No	No
Reuters Marketfeed Output	External	No	No	No
Reuters OMM Input	External	No	No	No
Reuters OMM Output	External	No	No	No
RTView Output	External	No	No	No
Tibco Rendezvous Input	External	Yes	No	Yes
Tibco Rendezvous Output	External	Yes	No	Yes

Editing Adapter Property Sets

Use the CCR Project Configuration editor in Studio to configure adapter property sets. Property sets are reusable sets of properties that are stored in the project configuration file.

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

1. In the CCR Project Configuration editor, select the **Adapter Properties** tab.
2. To create a new adapter property node, click **Add**.

3. Define a name for the property node in the **Property Set Details** pane.
4. 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.

5. To configure a property:
 - a) Define a name for the property in the **Property Details** pane.
 - b) Enter a value for the property.
6. (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**.

7. 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**.

Internal Adapters

Event Stream Processor provides internal adapters for processing standard data formats.

Internal adapters execute within Event Stream Processor. The Server starts and stops these adapters with query module execution.

See also

- *Adapter Parameters Datatypes* on page 8
- *Adapters that Support Schema Discovery* on page 539

AtomReader Input Adapter

Adapter type: atomreader_in. The AtomReader Input adapter allows you to receive information from ATOM datasources.

ATOM datasources enable connections through URLs and transmit their information in a specialized XML format. Information the adapter receives from an ATOM datasource is inserted into an Event Stream Processor stream.

Ensure that incoming XML information includes:

- feed_title
- feed_link

- feed_author_name
- entry_title
- entry_link
- entry_content

Note: The adapter ignores any additional fields in the XML file.

If you are not already familiar with the specific XML format ATOM uses, see: <http://www.atomenabled.org/>

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `atomreader_in`.

Property Label	Property ID	Type	Description
Source URL	URL	string	(Required) The URL of the ATOM datasource.
Refresh interval	refreshInterval	interval	(Optional) Determines how often the specified URL is queried for data. The adapter measures the interval in microseconds unless qualified with interval formatting. Default value is 60000 milliseconds.
Timestamp format	timestampFormat	string	(Advanced) The format string for parsing timestamp values. Default value is %Y-%m-%dT%H:%M:%S.
Date Format	dateFormat	string	(Advanced) The format string for parsing date values. Default value is %Y-%m-%dT%H:%M:%S.

Property Label	Property ID	Type	Description
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.

Database Adapter

Event Stream Processor provides an input and output database adapter. The Database Input adapter receives data from a database table, and the Database Output adapter sends data to a database table.

You can use several different JDBC and ODBC drivers with the database adapter. To use the ODBC drivers, ensure you have a driver manager installed.

Database Input Adapter

Adapter type: db_in. The Database Input adapter receives data from a database table.

You can use the adapter to periodically poll the table and receive updates. The required properties depend on the database type you are connecting to. The supported databases for JDBC are Adaptive Server® Enterprise, Microsoft SQL Server, IBM DB2, Oracle, and KDB. The supported databases for ODBC are Adaptive Server Enterprise, Microsoft SQL Server, IBM DB2, Oracle, Sybase IQ, SQL Anywhere®, TimesTen, MySQL 5.x, and PostgreSQL.

The `service.xml` file contains service definitions and the properties required for a database connection. For the service definition name, consult the person responsible for configuring and maintaining the `service.xml` file. See the *Administrators Guide* for information on configuring database connections using the `service.xml` file.

Use the **query** property to override the table selection and get data from an arbitrary query. This adapter supports schema discovery.

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is db_in.

Important: For an ASE database, enable the "ddl in tran" option on the temporary database (tempdb) to discover all tables when using schema discovery. Then, update the Server by

performing a checkpoint on tempdb or restarting the database instance. For more information on the "ddl in tran" option, consult your Adaptive Server documentation.

Property Label	Property ID	Type	Description
Database Service	service	string	(Required) Name of database service as defined in the <code>service.xml</code> file. No default value.
Database Query	query	string	(Optional) The SQL query to be executed by the adapter. No default value. Note: The adapter definition requires either query or table to be defined. If both parameters are defined the query parameter is used.
Input Table Name	table	tables	(Optional) Name of the table to read. No default value.
Poll Period (in seconds)	pollperiod	uint	(Advanced) Period for polling for new contents, in seconds. Default value is 0, which means no polling.
Date Format	dateFormat	string	(Advanced) Format string for parsing date values. Default value is <code>%Y-%m-%dT%H:%M:%S</code> .
Timestamp Format	timestamp-Format	string	(Advanced) Format string for parsing timestamp values. Default value is <code>%Y-%m-%dT%H:%M:%S</code> .
Field Mapping	permutation	permutation	(Advanced) Mapping between the in-platform and external fields. No default value. Note: <ul style="list-style-type: none"> For Oracle 11g and DB2 9.7, the metadata services return results in uppercase, so ensure the database column key in the permutation is in uppercase. For example, in CCL, this command does not work: <pre>permutation= 'Subject=sub- subject:c_string=c_string'</pre> but this one does: <pre>permutation= 'Subject=SUB- JECT:c_string=C_STRING'</pre> For ASE 15.5 and SQL Server 2008, the metadata results are the same as the case in the defined column name, unless the name is modified in the SELECT statement.

Property Label	Property ID	Type	Description
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.

The Database Input adapter has these limitations:

- Ensure, when polling, that this is the only adapter.
- Any data updates received from any other source are undone on the next poll.

Database Output Adapter

Adapter type: db_out. The Database Output adapter sends data to a database table.

You can truncate the table when the adapter starts using the **truncateTable** property. The required properties depend on the database type you are connecting to. The supported databases for JDBC are Adaptive Server Enterprise, Microsoft SQL Server, IBM DB2, Oracle, and KDB. The supported databases for ODBC are Adaptive Server Enterprise, Microsoft SQL Server, IBM DB2, Oracle, Sybase IQ, SQL Anywhere, TimesTen, MySQL 5.x, and PostgreSQL.

The `service.xml` file contains service definitions and the properties required for a database connection. For the service definition name, consult the person responsible for configuring and maintaining the `service.xml` file. See the *Administrators Guide* for more information on configuring database connections using the `service.xml` file.

Attention: The Oracle ODBC driver does not support `SQL_C_SBIGINT/SQL_C_UBIGINT` parameters, causing errors when the Database Output adapter tries to write long and interval Event Stream Processor types to `bigint` type columns. To successfully use the Oracle and the TimesTen ODBC drivers with the Database Output adapter, add this parameter `<Parameter Name = "WriteBigIntAsChar" > true < /Parameter >` to the `service.xml` file.

An example of specifying a different date format is when inserting a date column into an Oracle Date column. The default Oracle date format is: `04-Apr-1964 17:12:00`, so you specify that the **dateFormat** parameter is `d-%b-%Y %H:%M:%S`.

Important: Enable the "Server side prepare" option in the ODBC configuration to ensure that the Database Output adapter writes successfully to the PostgreSQL database using the ODBC driver. For more information on this option, consult your ODBC documentation.

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `db_out`.

Important: For an ASE database, enable the "ddl in tran" option on the temporary database (tempdb) to discover all tables when using schema discovery. Then, update the Server by performing a checkpoint on tempdb or restarting the database instance. For more information on the "ddl in tran" option, consult your Adaptive Server documentation.

Property Label	Property ID	Type	Description
Database Service	service	string	(Required) Name of database service as defined in the <code>service.xml</code> file. No default value.
Date Format	dateFormat	string	(Advanced) Format string for parsing date values. Default value is <code>%Y-%m-%dT%H:%M:%S</code> .
Timestamp Format	timestampFormat	string	(Advanced) Format string for parsing timestamp values. Default value is <code>%Y-%m-%dT%H:%M:%S</code> .

Property Label	Property ID	Type	Description
Field Mapping	permutation	permutation	<p>(Advanced) Mapping between the in-platform and external fields. No default value.</p> <hr/> <p>Note:</p> <ul style="list-style-type: none"> For Oracle 11g and DB2 9.7, the metadata services return results in uppercase, so ensure the database column key in the permutation is in uppercase. For example, in CCL, this command does not work: <pre>permutation= 'Sub- ject=sub- ject:c_string=c_string'</pre> <p>but this one does:</p> <pre>permutation= 'Sub- ject=SUB- JECT:c_string=C_STRING'</pre> For ASE 15.5 and SQL Server 2008, the metadata results are the same as the case in the defined column name, unless the name is modified in the SELECT statement. <hr/>
Only Base Content	onlyBase	boolean	(Advanced) Send only the initial contents of the stream, once. Default value is false.

Property Label	Property ID	Type	Description
Batch Limit	batchLimit	uint	<p>(Advanced) Number of records to process as a batch. Default value is 1.</p> <hr/> <p>Note: Using UPSERT with batch processing may negatively impact performance, since this process may be terminated if the adapter receives a delete. The resolution of an UPSERT to either INSERT or UPDATE based on stream content gives you less control over the grouping of these operations. However, frequently changing between operations (INSERT, UPDATE, DELETE, and UPSERT) reduces the optimization of using batch processing.</p>
Data Location	datalocation	string	(Advanced) Looks up properties in the project configuration. No default value.
Output Table Name (runtime)	table	tables	(Optional) Name of the table to push data to. No default value.
Include Base Content	outputBase	boolean	(Optional) Output initial stream contents in addition to stream updates. Default value is false.
Truncate the Database Table	truncateTable	boolean	(Optional) Start by truncating the database table, then populating with streaming data. Default value is false.
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.

The Database Output adapter has these limitations:

- The output table must exist.

CHAPTER 2: Adapters Supported by Event Stream Processor

- Each row translates to an SQL statement, therefore updates are slow.
- If you are using a memory store, you can perform only UPSERT, UPDATE, and DELETE on data that is in the stream.

Datatype Mapping for the Database Adapter

Mapping between Event Stream Processor datatypes and Sybase Adaptive Service Enterprise, Microsoft SQL Server, IBM DB2, Oracle, and KDB datatypes.

Datatype Mapping: Sybase ASE

Mapping between Event Stream Processor datatypes and Adaptive Server Enterprise 15.5 datatypes.

Event Stream Processor Datatype	Adaptive Server Enterprise Datatype
integer	int
long	bigint
float	float
date	datetime
string	varchar(n)
money	money
timestamp	bigdatetime
boolean	smallint or bit (Does not support null values)
money1	numeric(19,1)
money2	numeric(19,2)
money3	numeric(19,3)
money4	numeric(19,4)
money5	numeric(19,5)
money6	numeric(19,6)
money7	numeric(19,7)
money8	numeric(19,8)
money9	numeric(19,9)

Event Stream Processor Datatype	Adaptive Server Enterprise Datatype
money10	numeric(19,10)
money11	numeric(19,11)
money12	numeric(19,12)
money13	numeric(19,13)
money14	numeric(19,14)
money15	numeric(19,15)
interval	bigint
bigdatetime	bigdatetime
binary	varbinary(n)

Datatype Mapping: Microsoft SQL Server Database

Mapping between Event Stream Processor datatypes and Microsoft SQL Server 2008 R2 datatypes.

Event Stream Processor Datatype	SQL Datatype
integer	int
long	bigint
float	float
date	datetime
string	varchar(n)
money	money
timestamp	datetime2
boolean	smallint or bit (Does not support null values)
money1	numeric(19,1)
money2	numeric(19,2)
money3	numeric(19,3)

Event Stream Processor Datatype	SQL Datatype
money4	numeric(19,4)
money5	numeric(19,5)
money6	numeric(19,6)
money7	numeric(19,7)
money8	numeric(19,8)
money9	numeric(19,9)
money10	numeric(19,10)
money11	numeric(19,11)
money12	numeric(19,12)
money13	numeric(19,13)
money14	numeric(19,14)
money15	numeric(19,15)
interval	bigint
bigdatetime	datetime2
binary	varbinary(n)

Datatype Mapping: IBM DB2 Database

Mapping between Event Stream Processor datatypes and IBM DB2 9.7 datatypes.

Event Stream Processor Datatype	IBM DB2 Datatype
integer	int
long	bigint
float	float
date	timestamp
string	varchar(n)
money	decimal(19,5)
timestamp	timestamp

Event Stream Processor Datatype	IBM DB2 Datatype
boolean	smallint or bit (Does not support null values)
money1	decimal(19,1)
money2	decimal(19,2)
money3	decimal(19,3)
money4	decimal(19,4)
money5	decimal(19,5)
money6	decimal(19,6)
money7	decimal(19,7)
money8	decimal(19,8)
money9	decimal(19,9)
money10	decimal(19,10)
money11	decimal(19,11)
money12	decimal(19,12)
money13	decimal(19,13)
money14	decimal(19,14)
money15	decimal(19,15)
interval	bigint
bigdatetime	timestamp
binary	blob

Datatype Mapping: Oracle Database

Mapping between Event Stream Processor datatypes and Oracle 11g datatypes.

Event Stream Processor Datatype	Oracle Datatype
integer	int

CHAPTER 2: Adapters Supported by Event Stream Processor

Event Stream Processor Datatype	Oracle Datatype
long	number (19)
float	float
date	date
string	varchar2(n)
money	number (19,4)
timestamp	timestamp
boolean	smallint or bit (Does not support null values)
money1	number (19,1)
money2	number (19,2)
money3	number (19,3)
money4	number (19,4)
money5	number (19,5)
money6	number (19,6)
money7	number (19,7)
money8	number (19,8)
money9	number (19,9)
money10	number (19,10)
money11	number (19,11)
money12	number (19,12)
money13	number (19,13)
money14	number (19,14)
money15	number (19,15)
interval	number (19)
bigdatetime	timestamp

Event Stream Processor Datatype	Oracle Datatype
binary	blob

Datatype Mapping: KDB Database

Mapping between Event Stream Processor datatypes and KDB datatypes.

Event Stream Processor Datatype	KDB Datatype
integer	int
long	long
float	float
date	datetime
string	symbol
money	–
timestamp	datetime
boolean	boolean (Does not support null)
money1	–
money2	–
money3	–
money4	–
money5	–
money6	–
money7	–
money8	–
money9	–
money10	–
money11	–
money12	–
money13	–

Event Stream Processor Datatype	KDB Datatype
money14	–
money15	–
interval	long
bigdatetime	–
binary	–

Note: Do not include Event Stream Processor datatypes in your output that do not have an equivalent KDB datatype.

File CSV Input Adapter

Adapter type: `dsv_in`. The File CSV Input adapter reads a file in Event Stream Processor delimited format.

Use this adapter to poll new data appended to the data file. The file does not require a header. If the file includes a header, it specifies the field names.

Sample record formats for the data file:

```
1. hasHeader=true
delimiter=,
expectStreamNameOpcode=false

Ts,ItemID,Price,Quantity,WarehouseZipCode,DeliveryZipCode
2004/06/17 10:00:00.000000,SKU1276532,50.00,1,10012,94086
2004/06/17 10:00:05.000000,SKU6723143,23.00,2,10012,94043

2. expectStreamNameOpcode=true
delimiter=,

Trades_in,i,2004/06/17
10:00:00.000000,SKU1276532,50.00,1,10012,94086
Trades_in,i,2004/06/17
10:00:05.000000,SKU6723143,23.00,2,10012,94043

3. expectStreamNameOpcode=false
timestampFormat=%Y/%m/%d %H:%M:%S
delimiter=,

2004/06/17 10:00:00.000000,SKU1276532,50.00,1,10012,94086
2004/06/17 10:00:05.000000,SKU6723143,23.00,2,10012,94043
```

This adapter supports schema discovery. If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `dsv_in`.

Property Label	Property ID	Type	Description
Directory	dir	directory	(Required) Specify the absolute path to the data files you want the adapter to read. For example, <user-name>/<folder name>. No default value.
File (in Directory)	file	tables	(Required) File to read. No default value.
Stream name, opcode expected	expectStreamNameOpcode	boolean	(Optional) If true, the adapter interprets the first two fields as stream name and opcode respectively. Messages with unmatched stream names are discarded. Default value is false.
Field Count	fieldCount	uint	(Optional) Count of fields in CSV file, if different from the value for the source stream. Default value is 0.
Repeat Count	repeatCount	uint	(Optional) Number of times the input data is repeated. If set to -1, the input data is repeated indefinitely. Default value is 0. Note: This parameter can be used for testing a continuous streaming source.
Repeat Field	repeatField	string	(Optional) Determines which numeric field's values are bumped on each repeat. Default value is a hyphen (-). Note: <ul style="list-style-type: none"> • If repeatCount has a nonzero value, specify the stream column name. • If the repeatColumn is a key column in the stream, ensure there are no duplicates when specifying multiple rows in the input file. • If the adapter is attached to a window, the repeatField must be a key column.

Property Label	Property ID	Type	Description
Delimiter	delimiter	string	(Advanced) Symbol used to separate the column. Default value is a comma (,).
Has Header	hasHeader	boolean	(Advanced) Determines whether the first line of the file contains the description of the fields. Default value is false.
Directory (runtime)	runtimeDir	runtimeDirectory	(Advanced) Location of the data files at runtime, if the value is different from the location defined at discovery time. No default value.
File Pattern	filePattern	string	(Advanced) Pattern used to look up files for discovery. Default value is *.csv.
Poll Period (seconds)	pollperiod	uint	(Advanced) Period for polling for new contents, in seconds. If set to 0, the File CSV Input adapter will not poll the file for appended records. Default value is 0.
Convert to Safe Opcodes	safeOps	boolean	(Advanced) Converts the opcodes INSERT and UPDATE to UPSERT, and converts DELETE to SAFEDELETE. Default value is false.
Skip Deletes	skipDels	boolean	(Advanced) Skips the rows with opcodes DELETE or SAFEDELETE. Default value is false.
Date Format	dateFormat	string	(Advanced) Format string for parsing date values. Default value is %Y-%m-%dT%H:%M:%S.
Timestamp Format	timestampFormat	string	(Advanced) Format string for parsing timestamp values. Default value is %Y-%m-%dT%H:%M:%S.
Block Size	blockSize	int	(Advanced) Number of records to block into one pseudotransaction. Default value is 1.

Property Label	Property ID	Type	Description
Field Mapping	permutation	permutation	(Advanced) Mapping between the in-platform and external fields. No default value.
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.

The File CSV Input adapter has these limitations:

- When polling, you can append to the file, but the file cannot be overwritten or replaced. The stream names in the file rows are ignored and all the data is sent to the same stream.
- For discovery to work correctly, set the delimiter character and the header presence flag to match the actual data.
- Do not mix files with different delimiters or files with and without headers in the same directory. Files with wrong delimiters or headers are incorrectly discovered.

File CSV Output Adapter

Adapter type: `dsv_out`. The File CSV Output adapter writes data as a file in Event Stream Processor delimited format.

The file does not require a header. If the file includes a header, it specifies the field names. This adapter does not support schema discovery.

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `dsv_out`.

Property Label	Property ID	Type	Description
Directory	dir	directory	(Required) Specify the absolute path to the data files you want the adapter to read. For example, <code><username>/<folder name></code> . No default value.

Property Label	Property ID	Type	Description
File (in Directory)	file	tables	(Required) File the adapter writes data to. No default value.
Include Base Content	outputBase	boolean	(Optional) Records the initial contents of the stream, not just the updates. Default value is false.
Only Base Content	onlyBase	boolean	(Optional) Sends a one-time snapshot of initial contents in a stream. Default value is false.
Prepend StreamNameOpcode	prependStreamNameOpcode	boolean	(Optional) If true, each merge starts the stream name and the opcode. Default value is false.
Delimiter	delimiter	string	(Advanced) Symbol used to separate the columns. Default value is a comma (,).
Has Header	hasHeader	boolean	(Advanced) Determines whether the first line of the file contains the description of the fields. Default value is false.
Directory (runtime)	runtimeDir	runtimeDirectory	(Advanced) Location of the data files at runtime, if different from discovery time. No default value.
File Pattern	filePattern	string	(Advanced) Pattern used to look up files for discovery. Default value is *.csv.
Date Format	dateFormat	string	(Advanced) Format string to parse data values. Default value is %Y-%m-%dT%H:%M:%S.

Property Label	Property ID	Type	Description
Timestamp Format	timestampFormat	string	(Advanced) Format string to parse timestamp values. Default value is %Y-%m-%dT%H:%M:%S.
Field Mapping	permutation	permutation	(Advanced) Mapping between the in-platform and external fields. No default value.
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.

File XML Input Adapter

Adapter type: `xml_in`. The File XML Input adapter reads a file in XML format.

This adapter polls for new data appended to a file, and supports schema discovery.

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `xml_in`.

Sample record format for the data file:

```
<Trades Id="0" Symbol="EBAY" TradeTime="2000-05-04T12:00:00"
Price="140.0" Shares="50" />
<Trades Id="1" Symbol="EBAY" TradeTime="2000-05-04T12:00:01"
Price="150.0" Shares="500" />
```

Property Label	Property ID	Type	Description
Directory	dir	directory	(Required) Specify the absolute path to the data files you want the adapter to read. For example, <username>/<folder name>. No default value.
File (in Directory)	file	tables	(Required) File the adapter writes data to. No default value.
Match Stream Name	matchStreamName	boolean	(Optional) If true, XML element name is matched against the stream name. Unmatched messages are discarded. Default value is false.
Repeat Count	repeatCount	uint	(Optional) Number of times the input data is repeated. If set to -1, the input data is repeated indefinitely. Default value is 0. Note: This parameter can be used for testing a continuous streaming source.

Property Label	Property ID	Type	Description
Repeat Field	repeatField	string	<p>(Optional) Determines which numeric field's values are bumped on each repeat. Default value is a hyphen (-).</p> <hr/> <p>Note:</p> <ul style="list-style-type: none"> • If repeatCount has a nonzero value, specify the stream column name. • If the repeatColumn is a key column in the stream, ensure there are no duplicates when specifying multiple rows in the input file. • If the adapter is attached to a window, the repeatField must be a key column.
Directory (run-time)	runtimeDir	runtime-Directory	(Advanced) Location of the data files at run time, if different from discovery time. No default value.
File Pattern	filePattern	string	(Advanced) Pattern used to look up files for discovery. Default value is *.xml.
Poll Period (seconds)	pollperiod	uint	(Advanced) Period for polling new contents, in seconds. Default value is 0.
Convert to Safe Opcodes	safeOps	boolean	(Advanced) Converts the opcodes INSERT and UPDATE to UPSERT. Converts DELETE to SAFEDELETE. Default value is false.
Skip Deletes	skipDels	boolean	(Advanced) Skips the rows with opcodes DELETE or SAFEDELETE. Default value is false.

Property Label	Property ID	Type	Description
Date Format	dateFormat	string	(Advanced) Format string for parsing data values. Default value is %Y-%m-%dT%H:%M:%S.
Timestamp Format	timestampFormat	string	(Advanced) Format string for parsing timestamp values. Default value is %Y-%m-%dT%H:%M:%S.
Block Size	blockSize	int	(Advanced) Determines the number of records to block into one pseudotransaction. Default value is 1.
Field Mapping	permutation	permutation	(Advanced) Mapping between the in-platform and external fields. No default value.
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.

Known limitations:

- When polling, you can append to a file, but cannot overwrite or replace the file.
- The stream name in the file entries is ignored.
- Do not mix data files and model XML files in the same directory. This causes Event Stream Processor XML files to be discovered as invalid.

File XML Output Adapter

Adapter type: `xml_out`. The File XML Output adapter writes data as a file in XML format.

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `xml_out`.

Property Label	Property ID	Type	Description
Directory	dir	directory	(Required) Specify the absolute path to the data files you want the adapter to read. For example, <code><username>/<folder name></code> . No default value.
File (in Directory)	file	tables	(Required) File the adapter writes data to. No default value.
Include Base Content	output-Base	boolean	(Optional) Records the initial contents of the stream, not just the updates. Default value is false.
Only Base Content	onlyBase	boolean	(Optional) Sends a one-time snapshot of initial contents of the stream. Default value is false.
Directory (run-time)	runtimeDir	runtimeDirectory	(Advanced) Location of the data files at run time, if different from discovery time. No default value.
File Pattern	filePattern	string	(Advanced) Pattern used to look up files for discovery. Default value is <code>*.xml</code> .
Date Format	dateFormat	string	(Advanced) Format string for parsing data values. Default value is <code>%Y-%m-%dT%H:%M:%S</code> .
Timestamp Format	timestampFormat	string	(Advanced) Format string for parsing timestamp values. Default value is <code>%Y-%m-%dT%H:%M:%S</code> .
Field Mapping	permutation	permutation	(Advanced) Mapping between the in-platform and external fields. No default value.

Property Label	Property ID	Type	Description
PropertySet	property-set	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.

File FIX Input Adapter

Adapter type: `fixfile_in`. The File FIX Input adapter reads FIX messages from a file and writes them as stream records.

Each stream hosts FIX messages of a certain type. The adapter ignores messages of any other FIX type. It writes all FIX fields, except the following, in the same order in stream columns:

- BeginString
- BodyLength
- MessageType
- CheckSum

Ensure the names of the stream columns correspond to the FIX protocol specification.

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `fixfile_in`.

Property Label	Property ID	Type	Description
FIX Version	fixVersion	choice	(Required) Version of the FIX protocol. Default value is 4.2.
FIX Message Type	fixMessageType	string	(Required) Type of messages hosted by the stream. No default value.
File	fileName	filename	(Required) Path to the input file containing FIX messages. No default value.

Property Label	Property ID	Type	Description
Date Format	dateFormat	string	(Advanced) Date format. Default value is YYYY-MM-DDTHH:MM:SS.SSS.
Timestamp Format	timestampFormat	string	(Advanced) Timestamp format. Default value is YYYY-MM-DDTHH:MM:SS.SSS.
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.

Known limitations:

- This adapter is not a full FIX Engine.
- Supports only FIX versions 4.2, 4.3, 4.4, and 5.0.
- Does not support repeating groups and components.
- Supports only INSERT opcode.

See also

- *FIX Adapter* on page 135

Datatype Mapping for the File FIX Input Adapter

Event Stream Processor datatypes map to FIX datatypes.

Event Stream Processor Datatype	QuickFix Datatype
integer	boolean
string	byte[]
string	char
string	string

Event Stream Processor Datatype	QuickFix Datatype
date	date
float	float
integer	integer
date or timestamp	UTCDateOnly
date or timestamp	UTCTimeOnly
date or timestamp	UTCTimeStamp

File FIX Output Adapter

Adapter type: `fixfile_out`. The File FIX Output adapter writes stream data as FIX messages to a file.

Each stream hosts FIX messages of a certain type. The adapter writes messages to file in an adjoining manner, with no line feeds. It generates the following FIX fields:

- BeginString
- BodyLength
- MsgType
- CheckSum

Write the remaining fields in appropriate order in stream columns. Ensure the names of the stream columns correspond to the FIX protocol specification.

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `fixfile_out`.

Property Label	Property ID	Type	Description
FIX Version	fixVersion	choice	(Required) Version of the FIX protocol. Default value is 4.2.
FIX Message Type	fixMessageType	string	(Required) Type of messages hosted by the stream. No default value.
File	fileName	file-name	(Required) Path to the input file containing FIX messages. No default value.

Property Label	Property ID	Type	Description
Date Format	dateFormat	string	(Advanced) Date format. Default value is YYYY-MM-DDTHH:MM:SS.SSS.
Timestamp Format	timestampFormat	string	(Advanced) Timestamp format. Default value is YYYY-MM-DDTHH:MM:SS.SSS.
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.

Known limitations:

- This adapter is not a full FIX Engine.
- Supports only FIX versions 4.2, 4.3, 4.4, and 5.0.
- Does not support repeating groups and components.
- Does not support schema discovery.
- Supports only INSERT opcode.

See also

- *FIX Adapter* on page 135

Datatype Mapping for the File FIX Output Adapter

Event Stream Processor datatypes map to FIX datatypes.

Event Stream Processor Datatype	QuickFix Datatype
integer	boolean
string	byte[]
string	char

Event Stream Processor Datatype	QuickFix Datatype
string	string
date	date
float	float
integer	integer
date or timestamp	UTCDateOnly
date or timestamp	UTCTimeOnly
date or timestamp	UTCTimeStamp

JMS Adapter

Event Stream Processor supports five JMS Input and Output adapters: CSV, Custom, FIX, Object Array, and XML.

Configuring a Queuing System for JMS Adapter

To use the JMS adapters, configure the queuing system with the naming server.

Prerequisites

Queuing systems that support JNDI naming Servers.

Task

1. Set up a naming Server.
Some queuing systems contain internal naming Servers that could connect to JMS adapters.
2. Set up the queuing system to use the naming Server to administer JMS objects.
For more information, consult the documentation provided with your third-party queuing system.
3. Obtain the JNDI context library and the URL to get to the naming Server.

Note: To use the Sybase Event Stream Processor Adapter for JMS to integrate or communicate with TIBCO Enterprise Message Services, you must have a valid license for TIBCO Enterprise Message Services from TIBCO or from an authorized TIBCO channel.

For example, for Apache Active MQ, these are
`org.apache.activemq.jndi.ActiveMQInitialContextFactory` and
`tcp://localhost:61616`.

4. Set the `jndiContextFactory` and `jndiURL` properties for the JMS adapter.

5. Obtain the name that the JMS connection factory is bound by.
6. Set the **connectionFactory** property to this name for the adapter.
7. Ensure that appropriate JNDI and JMS `factory` classes are in the Java class path.

JMS CSV Input Adapter

Adapter type: `jms_csv_in`. The JMS CSV Input adapter subscribes to text messages formatted as a delimited list of values, and writes them as stream records.

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `jms_csv_in`.

When the `delimiter` is set to a comma (,) and the `expectStreamNameOpcode` is set to true, the JMS CSV Input adapter expects the input stream to be formatted as:

```
aaa,
11,111,1.100000,2008-03-13T08:19:30,111.1111,2008-03-13T08:19:30.12
3,false,FF00FE05FF,
2008-03-13T08:19:30.123456,64000,922.0,337.0000000000000000
```

The stream contains the following columns:

- `stringCol`
- `int32Col`
- `int64Col`
- `doubleCol`
- `dateCol`
- `moneyCol`
- `timestampCol`
- `booleanCol`
- `binaryCol`
- `bigdatetimeCol`
- `intervalCol`
- `money1Col`
- `money15Col`

```
<RecordType name="StreamIn_rec">
  <Column datatype="string" key="true" name="stringCol" />
  <Column datatype="integer" key="false" name="int32Col" />
  <Column datatype="long" key="false" name="int64Col" />
  <Column datatype="float" key="false" name="doubleCol" />
  <Column datatype="date" key="false" name="dateCol" />
  <Column datatype="money" key="false" name="moneyCol" />
  <Column datatype="timestamp" key="false" name="timestampCol" />
  <Column datatype="boolean" key="false" name="booleanCol" />
  <Column datatype="binary" key="false" name="binaryCol" />
  <Column datatype="bigdatetime" key="false"
name="bigdatetimeCol" />
  <Column datatype="interval" key="false" name="intervalCol" />
  <Column datatype="money(1)" key="false" name="money1Col" />
  <Column datatype="money(15)" key="false" name="money15Col" />
```

```
>
</RecordType>
```

Property Label	Property ID	Type	Description
Delimiter	delimiter	string	(Required) Field delimiter. Default value is a comma (.).
Connection Factory	connectionFactory	string	(Required) Connection factory class name. No default value.
JNDI Context Factory	jndiContextFactory	string	(Required) Context Factory for JNDI context initialization. No default value.
JNDI URL	jndiURL	string	(Required) JNDI URL. No default value.
Destination Type	destinationType	choice	(Required) Destination type. Valid values are: <ul style="list-style-type: none"> • QUEUE • TOPIC Default value is QUEUE.
Destination Name	destinationName	string	(Required) Destination name. No default value.
Subscription Mode	subscriptionMode	choice	(Optional) Specifies the subscription mode for TOPIC. Default value is NONDURABLE. Valid values are DURABLE and NONDURABLE.
Client ID	clientID	string	(Optional) Specifies the client identifier for the connection that is identifying durable subscription. No default value.

Property Label	Property ID	Type	Description
Subscription Name	subscriptionName	string	(Optional) Specifies a unique name identifying a durable subscription. No default value.
Batch Size	batchsize	uint	(Optional) Specifies number of records in a batch to commit in durable subscription mode. Default value is 1.
Stream Name Opcode Expected	expectStreamNameOpcode	boolean	(Advanced) If true, the first two fields in CSV records are interpreted as stream name, and opcode. Default value is false. <hr/> Note: An empty string is a valid value.
Date Format	dateFormat	string	(Advanced) Date format. Default value is YYYY-MM-DDTHH:MM:SS.SSS.
Timestamp Format	timestampFormat	string	(Advanced) Timestamp format. Default value is YYYY-MM-DDTHH:MM:SS.SSS.

Property Label	Property ID	Type	Description
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.

Known limitations:

- If the connection to the message broker is lost, the adapter does not attempt to reconnect.

See also

- *Chapter 5, Guaranteed Delivery* on page 543

JMS CSV Output Adapter

Adapter type: `jms_csv_out`. The JMS CSV Output adapter publishes stream data as text messages formatted as a delimited list of values to a JMS queue or topic.

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `jms_csv_out`.

Property Label	Property ID	Type	Description
Delimiter	delimiter	string	(Required) Field delimiter. Default value is a comma (.).
Connection Factory	connectionFactory	string	(Required) Connection factory class name. No default value.

Property Label	Property ID	Type	Description
JNDI Context Factory	jndiContextFactory	string	(Required) Context Factory for JNDI context initialization. No default value.
JNDI URL	jndiURL	string	(Required) JNDI URL. No default value.
Destination Type	destinationType	choice	(Required) Destination type. Valid values are: <ul style="list-style-type: none"> • QUEUE • TOPIC Default value is QUEUE.
Destination Name	destinationName	string	(Required) Destination name. No default value.
Delivery Mode	deliveryMode	choice	(Optional) Type of delivery mode. Valid values are: <ul style="list-style-type: none"> • PERSISTENT • NON_PERSISTENT Default value is PERSISTENT.

Property Label	Property ID	Type	Description
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.
Column To Message Property Map	columnPropertyMap	string	(Advanced) A delimited list of ColumnName=PropertyName mappings that enables message filtering on the message broker side using the JMS selector mechanism. For each mapped column name, the outbound message is paired with a corresponding JMS property that has a value equal to the column value. ColumnName1=PropertyName1, ColumnName2=PropertyName2 . . . No default value. Note: Ensure that there are no spaces in the value of this property.

Property Label	Property ID	Type	Description
Prepend Stream Name, Opcode	prependStreamNameOpcode	boolean	(Advanced) If true, every CSV record is prepended with stream name, and opcode. No default value.
Date Format	dateFormat	string	(Advanced) Date format. Default value is YYYY-MM-DDTHH:MM:SS.SSS.
Timestamp Format	timestampFormat	string	(Advanced) Timestamp format. Default value is YYYY-MM-DDTHH:MM:SS.SSS.
Runs Adapter in GD Mode	enableGDMode	boolean	(Advanced) If set to true, the adapter runs in guaranteed delivery (GD) mode and all GD-related parameters become required. Default value is false.
Name of Column Holding GD Key	gdKeyColumn	string	(Advanced) Specifies column name in the Flex operator holding the GD key. The GD key is a constantly increasing value that uniquely identifies every event regardless of the opcode in the stream of interest. No default value.

Property Label	Property ID	Type	Description
Name of Column Holding opcode	gdOpcodeColumn	string	(Advanced) Specifies name of column in Flex operator holding opcode. The opcode is the operation code (for example, inserts, update, or delete) of the event occurring in the stream of interest. No default value.
Name of Truncate Stream	gdControlStream	string	(Advanced) Specifies name of the control window in the GD setup. The control window is a source stream that informs the Flex operator of which data has been processed by the adapter and can be safely deleted. No default value.
Purge After Number of Records	gdPurgeInternal	int	(Advanced) Specifies number of records after which to purge the Flex operator. Default value is 1000.
Batch Size to Update Truncate Stream	gdBatchSize	int	(Advanced) Specifies number of records after which the control window must be updated with the latest GD key. Default value is 1000.

Known limitations:

- If the connection to the message broker is lost, the adapter does not attempt to reconnect.

See also

- *Chapter 5, Guaranteed Delivery* on page 543

JMS Custom Input Adapter

Adapter type: `jms_custom_in`. The JMS Custom Input adapter subscribes to custom-formatted Java object messages from a JMS queue or topic, and writes them as stream records.

A custom-provided implementation performs the format conversions of this interface:

```
package com.sybase.esp.adapters;
public interface ExternalToESPConverter {
    public ESPMessage externalToESP(Serializable externalMessage)
    throws Exception;
}
```

Ensure that the objects returned by the `externalToESP` method implement this interface:

```
package com.sybase.esp.adapters;
public interface ESPMessage extends Serializable {
    public String getStreamName();
    public String getOpCode();
    public Map<String, Serializable> getColumnValues();
}
```

The objects returned by the `getStreamName`, `getOpCode`, `getColumnValues` methods are interpreted as the name of the stream to write to, the opcode, and the stream record as a column to message property map value.

Ensure that stream column types correspond to Java classes as follows:

Stream Column Type	Java Class
<code>bigdatetime</code>	<code>java.lang.Double</code>
<code>binary</code>	<code>java.lang.String</code>
<code>boolean</code>	<code>java.lang.Boolean</code>
<code>integer</code>	<code>java.lang.Integer</code>
<code>interval</code>	<code>java.lang.Long</code>
<code>date</code>	<code>java.util.Date</code>
<code>float</code>	<code>java.lang.Double</code>
<code>long</code>	<code>java.lang.Long</code>
<code>money1</code>	<code>java.math.BigDecimal</code>
<code>money2</code>	<code>java.math.BigDecimal</code>
<code>money3</code>	<code>java.math.BigDecimal</code>
<code>money4</code>	<code>java.math.BigDecimal</code>

Stream Column Type	Java Class
money5	java.math.BigDecimal
money6	java.math.BigDecimal
money7	java.math.BigDecimal
money8	java.math.BigDecimal
money9	java.math.BigDecimal
money10	java.math.BigDecimal
money11	java.math.BigDecimal
money12	java.math.BigDecimal
money13	java.math.BigDecimal
money14	java.math.BigDecimal
money15	java.math.BigDecimal
string	java.lang.String
timestamp	java.util.Date

Ensure that implementations of the ExternalToEFSCConverter interface provide a constructor with a single argument of `java.lang.String` type, or a default constructor with no arguments.

Note: Records with unmatched stream names are ignored. Records with null opcodes are interpreted as upserts. The values of non-key columns may be absent or null.

If an implementation is not provided, the default implementation is used instead, which interprets each external message as an instance of the DefaultEFSCMessage class and does not perform a conversion.

Ensure that a Java archive containing an implementation of the ExternalToEFSCConverter interface is provided, and place it in the `lib` subfolder of the Event Stream Processor installation folder.

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `jms_custom_in`.

Note: This adapter supports schema discovery.

Property Label	Property ID	Type	Description
Converter Class Name	converterClassName	string	(Required) External to ESP message converter fully qualified class name. No default value.
Connection Factory	connectionFactory	string	(Required) Connection factory class name. No default value.
JNDI Context Factory	jndiContextFactory	string	(Required) Context factory for JNDI context initialization. No default value.
JNDI URL	jndiURL	string	(Required) JNDI URL. No default value.
Destination Type	destinationType	choice	(Required) Destination type. Valid values are: <ul style="list-style-type: none"> • QUEUE • TOPIC Default value is QUEUE.
Destination Name	destinationName	string	(Required) Destination name. No default value.
Converter Parameter	converterParam	string	(Optional) External to Event Stream Processor message converter start-up parameter. No default value.
Subscription Mode	subscriptionMode	choice	(Optional) Specifies the subscription mode for TOPIC. Default value is NONDURABLE. Valid values are DURABLE and NON-DURABLE.

Property Label	Property ID	Type	Description
Client ID	clientID	string	(Optional) Specifies the client identifier for the connection that is identifying durable subscription. No default value.
Subscription Name	subscriptionName	string	(Optional) Specifies a unique name identifying a durable subscription. No default value.
Batch Size	batchsize	uint	(Optional) Specifies number of records in a batch to commit in durable subscription mode. Default value is 1.
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.
Date Format	dateFormat	string	(Advanced) Date format. Default value is YYYY-MM-DDTHH:MM:SS.SSS.

Property Label	Property ID	Type	Description
Timestamp Format	<code>timestampFormat</code>	string	(Advanced) Timestamp format. Default value is YYYY-MM-DDTHH:MM:SS.SSS.

Known limitations:

- If the connection to the message broker is lost, the adapter does not attempt to reconnect.

See also

- *Chapter 5, Guaranteed Delivery* on page 543

JMS Custom Output Adapter

Adapter type: `jms_custom_out`. The JMS Custom Output adapter publishes stream records as custom-formatted Java objects to a JMS queue or topic.

A custom-provided implementation performs the format conversions of this interface:

```
package com.sybase.esp.adapters;
public interface ESPToExternalConverter {
    public Serializable ESPToExternal(ESPMessage ESPMessage) throws
Exception;
}
```

Ensure that stream column types correspond to Java classes as follows:

Stream Column Type	Java Class
<code>bigdatetime</code>	<code>java.lang.Double</code>
<code>binary</code>	<code>java.lang.String</code>
<code>boolean</code>	<code>java.lang.Boolean</code>
<code>integer</code>	<code>java.lang.Integer</code>
<code>interval</code>	<code>java.lang.Long</code>
<code>date</code>	<code>java.util.Date</code>
<code>float</code>	<code>java.lang.Double</code>
<code>long</code>	<code>java.lang.Long</code>
<code>money1</code>	<code>java.math.BigDecimal</code>
<code>money2</code>	<code>java.math.BigDecimal</code>

Stream Column Type	Java Class
money3	java.math.BigDecimal
money4	java.math.BigDecimal
money5	java.math.BigDecimal
money6	java.math.BigDecimal
money7	java.math.BigDecimal
money8	java.math.BigDecimal
money9	java.math.BigDecimal
money10	java.math.BigDecimal
money11	java.math.BigDecimal
money12	java.math.BigDecimal
money13	java.math.BigDecimal
money14	java.math.BigDecimal
money15	java.math.BigDecimal
string	java.lang.String
timestamp	java.util.Date

Ensure that implementations of the `ESPToExternalConverter` interface provide a constructor with a single argument of `java.lang.String` type or a default constructor with no arguments.

Note: The stream name, the opcode and the map of column name value of the `ESPMMessage` object are guaranteed to be valid, even if some non-key column values may be null.

Ensure that a Java archive containing an implementation of the `ExternalToEFSCConverter` interface is provided, and place it in the `lib` subfolder of the Event Stream Processor installation folder.

If an implementation is not provided, the default implementation is used and the `ESPMMessage` object is returned with no actual conversion performed.

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `jms_custom_out`.

Property Label	Property ID	Type	Description
Converter Class Name	converterClassName	string	(Required) External to Event Stream Processor message converter fully qualified class name. No default value.
Connection Factory	connectionFactory	string	(Required) Connection factory class name. No default value.
JNDI Context Factory	jndiContextFactory	string	(Required) Context Factory for JNDI context initialization. No default value.
JNDI URL	jndiURL	string	(Required) JNDI URL. No default value.
Destination Type	destinationType	choice	(Required) Destination type. Valid values are: <ul style="list-style-type: none"> • QUEUE • TOPIC Default value is QUEUE.
Destination Name	destinationName	string	(Required) Destination name. No default value.
Delivery Mode	deliveryMode	choice	(Optional) Type of delivery mode. Valid values are: <ul style="list-style-type: none"> • PERSISTENT • NON_PERSISTENT Default value is PERSISTENT.
Converter Parameter	converterParam	string	(Optional) External to ESP message converter startup parameter. No default value.

Property Label	Property ID	Type	Description
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.
Column To Message Property Map	columnPropertyMap	string	(Advanced) A comma-delimited list of ColumnName=PropertyName mappings which enables message filtering on the message broker side using the JMS selector mechanism. For each mapped column name, the outbound message is paired with a corresponding JMS property whose value equals the column value. ColumnName1=PropertyName1, ColumnName2=PropertyName2 . . . No default value. Note: Ensure that there are no spaces in the value of this property.
Date Format	dateFormat	string	(Advanced) Date format. Default value is YYYY-MM-DDTHH:MM:SS.SSS.
Timestamp Format	timestampFormat	string	(Advanced) Timestamp format. Default value is YYYY-MM-DDTHH:MM:SS.SSS.

Property Label	Property ID	Type	Description
Runs Adapter in GD Mode	enableGDMode	boolean	(Advanced) If set to true, the adapter runs in guaranteed delivery (GD) mode and all GD-related parameters become required. Default value is false.
Name of Column Holding GD Key	gdKeyColumn	string	(Advanced) Specifies column name in the Flex operator holding the GD key. The GD key is a constantly increasing value that uniquely identifies every event regardless of the opcode in the stream of interest. No default value.
Name of Column Holding opcode	gdOpcodeColumn	string	(Advanced) Specifies name of column in Flex operator holding opcode. The opcode is the operation code (for example, inserts, update, or delete) of the event occurring in the stream of interest. No default value.
Name of Truncate Stream	gdControlStream	string	(Advanced) Specifies name of the control window in the GD setup. The control window is a source stream that informs the Flex operator of which data has been processed by the adapter and can be safely deleted. No default value.
Purge After Number of Records	gdPurgeInternal	int	(Advanced) Specifies number of records after which to purge the Flex operator. Default value is 1000.
Batch Size to Update Truncate Stream	gdBatchSize	int	(Advanced) Specifies number of records after which the control window must be updated with the latest GD key. Default value is 1000.

Known limitations:

- If the connection to the message broker is lost, the adapter does not attempt to reconnect.

See also

- *Chapter 5, Guaranteed Delivery* on page 543

JMS FIX Input Adapter

Adapter type: `jms_fix_in`. The JMS FIX Input adapter subscribes to messages from a JMS queue or topic, and writes these messages as stream records.

Each stream hosts FIX messages of a certain type. The adapter discards messages of any other FIX type. Most FIX fields are stored in the same order in stream columns however these fields can be stored in a different order:

- `BeginString`
- `BodyLength`
- `MsgType`
- `Checksum`

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `jms_fix_in`.

Property Label	Property ID	Type	Description
FIX Version	fixVersion	choice	(Required) FIX Version. Valid values are: <ul style="list-style-type: none"> • 4.2 • 4.3 • 4.4 • 5.0 Default value is 4.2.
FIX Message Type	fixMessageType	string	(Required) FIX Message type. No default value.
Connection Factory	connectionFactory	string	(Required) Connection factory class name. No default value.
JNDI Context Factory	jndiContextFactory	string	(Required) Context Factory for JNDI context initialization. No default value.

Property Label	Property ID	Type	Description
JNDI URL	jndiURL	string	(Required) JNDI URL. No default value.
Destination Type	destinationType	choice	(Required) Destination type. Valid values are: <ul style="list-style-type: none"> • QUEUE • TOPIC Default value is QUEUE.
Destination Name	destinationName	string	(Required) Destination name. No default value.
Subscription Mode	subscriptionMode	choice	(Optional) Specifies the subscription mode for TOPIC. Default value is NONDURABLE. Valid values are DURABLE and NONDURABLE.
Client ID	clientID	string	(Optional) Specifies the client identifier for the connection that is identifying durable subscription. No default value.
Subscription Name	subscriptionName	string	(Optional) Specifies a unique name identifying a durable subscription. No default value.
Batch Size	batchsize	uint	(Optional) Specifies number of records in a batch to commit in durable subscription mode. Default value is 1.

Property Label	Property ID	Type	Description
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.
Date Format	dateFormat	string	(Advanced) Date format. Default value is YYYY-MM-DDTHH:MM:SS.SSS
Timestamp Format	timestampFormat	string	(Advanced) Timestamp format. Default value is YYYY-MM-DDTHH:MM:SS.SSS

Known limitations:

- This adapter is not a full FIX engine.
- Supports only FIX versions 4.2, 4.3, 4.4, and 5.0.
- Repeating groups and components are not supported.
- If the connection to the message broker is lost, the adapter does not attempt to reconnect.
- Supports only INSERT opcode.

See also

- *Chapter 5, Guaranteed Delivery* on page 543

JMS FIX Output Adapter

Adapter type: `jms_fix_out`. The JMS FIX Output adapter publishes FIX messages to a JMS queue or topic.

Each stream hosts FIX messages of a certain type. Messages of any other FIX type are discarded. All FIX fields except the following are stored in the same order in stream columns.

- BeginString
- BodyLength
- MsgType
- CheckSum

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `jms_fix_out`.

Property Label	Property ID	Type	Description
FIX Version	fixVersion	choice	(Required) FIX Version. Valid values are: <ul style="list-style-type: none"> • 4.2 • 4.3 • 4.4 • 5.0 Default value is 4.2.
FIX Message Type	fixMessageType	string	(Required) FIX Message type.
Connection Factory	connectionFactory	string	(Required) Connection factory class name.
JNDI Context Factory	jndiContextFactory	string	(Required) Context factory for JNDI context initialization.
JNDI URL	jndiURL	string	(Required) JNDI URL.
Destination Type	destinationType	choice	(Required) Destination type. Valid values are: <ul style="list-style-type: none"> • QUEUE • TOPIC Default value is QUEUE.
Destination Name	destinationName	string	(Required) Destination name.

Property Label	Property ID	Type	Description
Delivery Mode	deliveryMode	choice	(Optional) Type of delivery mode. Valid values are: <ul style="list-style-type: none"> • PERSISTENT • NON_PERSISTENT Default value is PERSISTENT.
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.
Column To Message Property Map	columnPropertyMap	string	(Advanced) Comma-delimited MyColumn=MyMessageProperty correspondence list
Date Format	dateFormat	string	(Advanced) Date format. Default value is YYYY-MM-DDTHH:MM:SS.SSS
Timestamp Format	timestampFormat	string	(Advanced) Timestamp format. Default value is YYYY-MM-DDTHH:MM:SS.SSS

Property Label	Property ID	Type	Description
Runs Adapter in GD Mode	enableGDMode	<code>boolean</code>	(Advanced) If set to true, the adapter runs in guaranteed delivery (GD) mode and all GD-related parameters become required. Default value is false.
Name of Column Holding GD Key	gdKeyColumn	<code>string</code>	(Advanced) Specifies column name in the Flex operator holding the GD key. The GD key is a constantly increasing value that uniquely identifies every event regardless of the opcode in the stream of interest. No default value.
Name of Column Holding opcode	gdOpcodeColumn	<code>string</code>	(Advanced) Specifies name of column in Flex operator holding opcode. The opcode is the operation code (for example, inserts, update, or delete) of the event occurring in the stream of interest. No default value.
Name of Truncate Stream	gdControlStream	<code>string</code>	(Advanced) Specifies name of the control window in the GD setup. The control window is a source stream that informs the Flex operator of which data has been processed by the adapter and can be safely deleted. No default value.

Property Label	Property ID	Type	Description
Purge After Number of Records	gdPurgeInternal	int	(Advanced) Specifies number of records after which to purge the Flex operator. Default value is 1000.
Batch Size to Update Truncate Stream	gdBatchSize	int	(Advanced) Specifies number of records after which the control window must be updated with the latest GD key. Default value is 1000.

Known limitations:

- This adapter is not a full FIX engine.
- Supports only FIX versions 4.2, 4.3, 4.4, and 5.0.
- Repeating groups and components are not supported.
- Schema discovery is not supported.
- If the connection to the message broker is lost, the adapter does not attempt to reconnect.
- Supports only INSERT opcode.

See also

- *Chapter 5, Guaranteed Delivery* on page 543

JMS Object Array Input Adapter

Adapter type: `jms_objarray_in`. The JMS Object Array Input adapter subscribes to messages formatted as arrays of Java objects from a JMS queue or topic, and writes these messages as stream records.

Note: A null element in the array generates a null value for the corresponding column.

Ensure that stream column types correspond to Java classes as follows:

Stream Column Type	Java Class
<code>bigdatetime</code>	<code>java.lang.Double</code>
<code>binary</code>	<code>java.lang.String</code>
<code>boolean</code>	<code>java.lang.Boolean</code>
<code>integer</code>	<code>java.lang.Integer</code>

Stream Column Type	Java Class
interval	java.lang.Long
date	java.util.Date
float	java.lang.Double
long	java.lang.Long
money1	java.math.BigDecimal
money2	java.math.BigDecimal
money3	java.math.BigDecimal
money4	java.math.BigDecimal
money5	java.math.BigDecimal
money6	java.math.BigDecimal
money7	java.math.BigDecimal
money8	java.math.BigDecimal
money9	java.math.BigDecimal
money10	java.math.BigDecimal
money11	java.math.BigDecimal
money12	java.math.BigDecimal
money13	java.math.BigDecimal
money14	java.math.BigDecimal
money15	java.math.BigDecimal
string	java.lang.String
timestamp	java.util.Date

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `jms_objarray_in`.

Property Label	Property ID	Type	Description
Connection Factory	connectionFactory	string	(Required) Connection factory class name. No default value.
JNDI Context Factory	jndiContextFactory	string	(Required) Context factory for JNDI context initialization. No default value.
JNDI URL	jndiURL	string	(Required) JNDI URL. No default value.
Destination Type	destinationType	choice	(Required) Destination type. Valid values are: <ul style="list-style-type: none"> • QUEUE • TOPIC Default value is QUEUE.
Destination Name	destinationName	string	(Required) Destination name. No default value.
Subscription Mode	subscriptionMode	choice	(Optional) Specifies the subscription mode for TOPIC. Default value is NONDURABLE. Valid values are DURABLE and NONDURABLE.
Client ID	clientID	string	(Optional) Specifies the client identifier for the connection that is identifying durable subscription. No default value.
Subscription Name	subscriptionName	string	(Optional) Specifies a unique name identifying a durable subscription. No default value.

Property Label	Property ID	Type	Description
Batch Size	batchsize	uint	(Optional) Specifies number of records in a batch to commit in durable subscription mode. Default value is 1.
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.
Stream Name Opcode Expected	expectStreamNameOpcode	boolean	(Advanced) If true, the first two fields in CSV records are interpreted as stream name, and opcode. Default value is false.
Date Format	dateFormat	string	(Advanced) Date format. Default value is YYYY-MM-DDTHH:MM:SS.SSS
Timestamp Format	timestampFormat	string	(Advanced) Timestamp format. Default value is YYYY-MM-DDTHH:MM:SS.SSS

Known limitations:

- If the connection to the message broker is lost, the adapter does not attempt to reconnect.

See also

- *Chapter 5, Guaranteed Delivery* on page 543

JMS Object Array Output Adapter

Adapter type: `jms_objarray_out`. The JMS Object Array Output adapter publishes stream data as an array of Java objects to a JMS queue or topic.

Note: A null value in the column generates a null element for the corresponding array.

Ensure that stream column types correspond to Java classes as follows:

Stream Column Type	Java Class
<code>bigdatetime</code>	<code>java.lang.Double</code>
<code>binary</code>	<code>java.lang.String</code>
<code>boolean</code>	<code>java.lang.Boolean</code>
<code>integer</code>	<code>java.lang.Integer</code>
<code>interval</code>	<code>java.lang.Long</code>
<code>date</code>	<code>java.util.Date</code>
<code>float</code>	<code>java.lang.Double</code>
<code>long</code>	<code>java.lang.Long</code>
<code>money1</code>	<code>java.math.BigDecimal</code>
<code>money2</code>	<code>java.math.BigDecimal</code>
<code>money3</code>	<code>java.math.BigDecimal</code>
<code>money4</code>	<code>java.math.BigDecimal</code>
<code>money5</code>	<code>java.math.BigDecimal</code>
<code>money6</code>	<code>java.math.BigDecimal</code>
<code>money7</code>	<code>java.math.BigDecimal</code>
<code>money8</code>	<code>java.math.BigDecimal</code>
<code>money9</code>	<code>java.math.BigDecimal</code>
<code>money10</code>	<code>java.math.BigDecimal</code>
<code>money11</code>	<code>java.math.BigDecimal</code>

Stream Column Type	Java Class
money12	java.math.BigDecimal
money13	java.math.BigDecimal
money14	java.math.BigDecimal
money15	java.math.BigDecimal
string	java.lang.String
timestamp	java.util.Date

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `jms_objarray_out`.

Property Label	Property ID	Type	Description
Connection Factory	connectionFactory	string	(Required) Connection factory class name. No default value.
JNDI Context Factory	jndiContextFactory	string	(Required) Context factory for JNDI context initialization. No default value.
JNDI URL	jndiURL	string	(Required) JNDI URL. No default value.
Destination Type	destinationType	choice	(Required) Destination type. Valid values are: <ul style="list-style-type: none"> • QUEUE • TOPIC Default value is QUEUE.
Destination Name	destinationName	string	(Required) Destination name.

Property Label	Property ID	Type	Description
Delivery Mode	deliveryMode	choice	<p>(Optional) Type of delivery mode.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> • PERSISTENT • NON_PERSISTENT <p>Default value is PERSISTENT.</p>
PropertySet	propertyset	string	<p>(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.</p>

Property Label	Property ID	Type	Description
Column To Message Property Map	columnPropertyMap	string	<p>(Advanced) A comma-delimited list of ColumnName=PropertyName mappings that enables message filtering on the message broker side using the JMS selector mechanism. For each mapped column name, the outbound message is paired with a corresponding JMS property whose value equals the column value. ColumnName1=PropertyName1, ColumnName2=PropertyName2...</p> <p>No default value.</p> <hr/> <p>Note: Ensure that no spaces are present in the value of this property.</p>
Prepend Stream Name, Opcode	prependStreamNameOpcode	boolean	(Advanced) If true, every CSV record is prepended with stream name, opcode. No default value.
Date Format	dateFormat	string	(Advanced) Date format. Default value is YYYY-MM-DDTHH:MM:SS.SSS
Timestamp Format	timestampFormat	string	(Advanced) Timestamp format. Default value is YYYY-MM-DDTHH:MM:SS.SSS

Property Label	Property ID	Type	Description
Runs Adapter in GD Mode	enableGDMode	boolean	(Advanced) If set to true, the adapter runs in guaranteed delivery (GD) mode and all GD-related parameters become required. Default value is false.
Name of Column Holding GD Key	gdKeyColumn	string	(Advanced) Specifies column name in the Flex operator holding the GD key. The GD key is a constantly increasing value that uniquely identifies every event regardless of the opcode in the stream of interest. No default value.
Name of Column Holding opcode	gdOpcodeColumn	string	(Advanced) Specifies name of column in Flex operator holding opcode. The opcode is the operation code (for example, inserts, update, or delete) of the event occurring in the stream of interest. No default value.
Name of Truncate Stream	gdControlStream	string	(Advanced) Specifies name of the control window in the GD setup. The control window is a source stream that informs the Flex operator of which data has been processed by the adapter and can be safely deleted. No default value.

Property Label	Property ID	Type	Description
Purge After Number of Records	gdPurgeInternal	int	(Advanced) Specifies number of records after which to purge the Flex operator. Default value is 1000.
Batch Size to Update Truncate Stream	gdBatchSize	int	(Advanced) Specifies number of records after which the control window must be updated with the latest GD key. Default value is 1000.

Known limitations:

- If the connection to the message broker is lost, the adapter does not attempt to reconnect.

See also

- *Chapter 5, Guaranteed Delivery* on page 543

JMS XML Input Adapter

Adapter type: `jms_xml_in`. The JMS XML Input adapter subscribes to XML-formatted text messages from a JMS queue or topic, and writes the messages as stream records.

Ensure that each message consists of an XML element. If opted, the element name corresponds to the stream name.

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `jms_xml_in`.

Sample record format for the data file:

```
< StreamOut ESP_OPS="u" stringCol="aaa" int32Col="22" int64Col="222"
doubleCol="2.200000" dateCol="2008-03-13T08:19:30"
moneyCol="222.2222" timestampCol="2008-03-13T08:19:30.123"
booleanCol="true" binaryCol="FF00FE05FF"
bigdatetimeCol="2008-03-13T08:19:30.123456" intervalCol="64000"
money1Col="922.0" money15Col="337.0000000000000000" />
```

The stream contains the following columns:

- `stringCol`
- `int32Col`
- `int64Col`
- `doubleCol`
- `dateCol`

CHAPTER 2: Adapters Supported by Event Stream Processor

- moneyCol
- timestampCol
- booleanCol
- binaryCol
- bigdatetimeCol
- intervalCol
- money1Col
- money15Col

```
<RecordType name="StreamIn_rec">
  <Column datatype="string" key="true" name="stringCol" />
  <Column datatype="integer" key="false" name="int32Col" />
  <Column datatype="long" key="false" name="int64Col" />
  <Column datatype="float" key="false" name="doubleCol" />
  <Column datatype="date" key="false" name="dateCol" />
  <Column datatype="money" key="false" name="moneyCol" />
  <Column datatype="timestamp" key="false" name="timestampCol" />
  <Column datatype="boolean" key="false" name="booleanCol" />
  <Column datatype="binary" key="false" name="binaryCol" />
  <Column datatype="bigdatetime" key="false"
name="bigdatetimeCol" />
  <Column datatype="interval" key="false" name="intervalCol" />
  <Column datatype="money(1)" key="false" name="money1Col" />
  <Column datatype="money(15)" key="false" name="money15Col" /
>
</RecordType>
```

The ESP-OPS attribute is optional. If omitted, the message is interpreted as an upsert. Ensure that the rest of the attributes have the same names as the corresponding stream columns, and that the columns with null values are omitted. This adapter supports schema discovery.

Property Label	Property ID	Type	Description
Connection Factory	connectionFactory	string	(Required) Connection factory class name. No default value.
JNDI Context Factory	jndiContextFactory	string	(Required) Context Factory for JNDI context initialization. No default value.
JNDI URL	jndiURL	string	(Required) JNDI URL. No default value.

Property Label	Property ID	Type	Description
Destination Type	destinationType	choice	(Required) Destination type. Valid values are: <ul style="list-style-type: none"> • QUEUE • TOPIC Default value is QUEUE.
Destination Name	destinationName	string	(Required) Destination name. No default value.
Subscription Mode	subscriptionMode	choice	(Optional) Specifies the subscription mode for TOPIC. Default value is NONDURABLE. Valid values are DURABLE and NONDURABLE.
Client ID	clientID	string	(Optional) Specifies the client identifier for the connection that is identifying durable subscription. No default value.
Subscription Name	subscriptionName	string	(Optional) Specifies a unique name identifying a durable subscription. No default value.
Batch Size	batchsize	uint	(Optional) Specifies number of records in a batch to commit in durable subscription mode. Default value is 1.
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.

Property Label	Property ID	Type	Description
Match Stream Name	matchStreamName	boolean	(Advanced) Ignore message if the XML element name does not match the source stream name. Default value is false.
Date Format	dateFormat	string	(Advanced) Date format. Default value is YYYY-MM-DDTHH:MM:SS.SSS
Timestamp Format	timestampFormat	string	(Advanced) Timestamp format. Default value is YYYY-MM-DDTHH:MM:SS.SSS

Known limitations:

- If the connection to the message broker is lost, the adapter does not attempt to reconnect.

See also

- *Chapter 5, Guaranteed Delivery* on page 543

JMS XML Output Adapter

Adapter type: `jms_xml_out`. The JMS XML Output adapter publishes stream data as XML-formatted text messages to a JMS queue or topic.

Ensure that each message consists of an XML element with the same name as the stream name.

The first attribute is the Event Stream Processor opcode. The rest of the attributes have the same names as the corresponding stream columns. Ensure that any columns with null values are omitted.

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `jms_xml_out`.

Property Label	Property ID	Type	Description
Connection Factory	connectionFactory	string	(Required) Connection factory class name. No default value.
JNDI Context Factory	jndiContextFactory	string	(Required) Context factory for JNDI context initialization. No default value.

Property Label	Property ID	Type	Description
JNDI URL	jndiURL	string	(Required) JNDI URL. No default value.
Destination Type	destinationType	choice	(Required) Destination type. Valid values are: <ul style="list-style-type: none"> • QUEUE • TOPIC Default value is QUEUE.
Destination Name	destinationName	string	(Required) Destination name. No default value.
Delivery Mode	deliveryMode	choice	(Optional) Type of delivery mode. Valid values are: <ul style="list-style-type: none"> • PERSISTENT • NON_PERSISTENT Default value is PERSISTENT.

Property Label	Property ID	Type	Description
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.

Property Label	Property ID	Type	Description
Column To Message Property Map	columnPropertyMap	string	<p>(Advanced) A comma-delimited list of ColumnName=PropertyName mappings that enables message filtering on the message broker side using the JMS selector mechanism. For each mapped column name, the outbound message is paired with a corresponding JMS property that has a value equal to the column value.</p> <p>ColumnName1=PropertyName1,ColumnName2=PropertyName2... No default value.</p> <hr/> <p>Note: Ensure that there are no spaces present in the value of this property.</p>
Date Format	dateFormat	string	(Advanced) Date format. Default value is YYYY-MM-DDTHH:MM:SS.SSS
Timestamp Format	timestampFormat	string	(Advanced) Timestamp format. Default value is YYYY-MM-DDTHH:MM:SS.SSS

Property Label	Property ID	Type	Description
Runs Adapter in GD Mode	enableGDMode	boolean	(Advanced) If set to true, the adapter runs in guaranteed delivery (GD) mode and all GD-related parameters become required. Default value is false.
Name of Column Holding GD Key	gdKeyColumn	string	(Advanced) Specifies column name in the Flex operator holding the GD key. The GD key is a constantly increasing value that uniquely identifies every event regardless of the opcode in the stream of interest. No default value.
Name of Column Holding opcode	gdOpcodeColumn	string	(Advanced) Specifies name of column in Flex operator holding opcode. The opcode is the operation code (for example, inserts, update, or delete) of the event occurring in the stream of interest. No default value.
Name of Truncate Stream	gdControlStream	string	(Advanced) Specifies name of the control window in the GD setup. The control window is a source stream that informs the Flex operator of which data has been processed by the adapter and can be safely deleted. No default value.

Property Label	Property ID	Type	Description
Purge After Number of Records	gdPurgeInternal	int	(Advanced) Specifies number of records after which to purge the Flex operator. Default value is 1000.
Batch Size to Update Truncate Stream	gdBatchSize	int	(Advanced) Specifies number of records after which the control window must be updated with the latest GD key. Default value is 1000.

Known limitations:

- If the connection to the message broker is lost, the adapter does not attempt to reconnect.

See also

- *Chapter 5, Guaranteed Delivery* on page 543

Random Tuples Generator Input Adapter

Adapter type: `randomtuplegen_in`. The Random Tuples Generator adapter generates random tuples according to the given schema and sends the rows to the stream.

A tuple is an ordered list of elements, or in other words, a row of data. A row that has two column values is a 2-tuple, and generally, a row that has n column values is an n-tuple. The adapter is primarily used for prototyping and basic testing of Event Stream Processor. You can edit both the schema and configuration file.

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `randomtuplegen_in`.

Property Label	Property ID	Type	Description
Rate	Rate	uint	<p>(Optional) Number of rows generated per second. Must be between 0 (exclusive) and 1,000,000 (inclusive).</p> <p>If Rate is negative, the adapter stops and returns a fatal error message to the Server.</p> <p>If Rate is larger than the maximum value, it resets to the maximum and reports a warning message to the Server.</p> <p>If Rate property is blank, it resets to the default value and reports a message.</p> <p>Default value is 100.</p>

Property Label	Property ID	Type	Description
Row Count	RowCount	uint	<p>(Optional) Specifies number of generated rows. Must be between 0 and 2,000,000,000 inclusive.</p> <p>If RowCount is negative, the adapter stops and returns a fatal error message to the Server.</p> <p>If RowCount is larger than the maximum value, it resets to the maximum and reports a warning message to the Server.</p> <p>If RowCount property is blank, it resets to the default value and reports an information message to the Server.</p> <p>Default value is 0, which represents an infinite number of rows.</p>

Property Label	Property ID	Type	Description
Timestamp Base	TimestampBase	string	<p>(Optional) Initial time for message time-stamps. The supported format of TimestampBase is %Y-%m-%dT%H:%M:%S.</p> <p>If TimestampBase is blank, it resets to default value and the adapter initializes immediately. Similarly, the adapter initializes immediately if the TimestampBase value is earlier than current time.</p> <p>If TimestampBase value is later than current time, the adapter sleeps until then.</p> <p>If TimestampBase has an invalid timestamp format, the adapter stops and returns a fatal error message to the Server.</p> <p>Default value is the current time.</p>
Date Format	DateFormat	string	<p>(Optional) Format string for parsing date values. Default value is %Y-%m-%d %H:%M:%S.</p>
Timestamp Format	TimestampFormat	string	<p>(Optional) Format string for parsing date values. Default value is %Y-%m-%d %H:%M:%S.</p>

Property Label	Property ID	Type	Description
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.

The data this adapter generates is not evenly distributed across the range of possible values for each datatype. This table shows the value range generated for each datatype:

Datatype	Range of values
boolean	true/false
integer	0 .. 99 inclusive
long	0 .. 99 inclusive
float	0.0 .. 10.0 exclusive (should never get 10.0)
interval	0 .. 9 inclusive
timestamp	Current time with milliseconds
string	2 characters from the following ranges a..z, A..Z, 0..9
binary	2 bytes each with range 0..255
money	0.0000 to 3.2767
money1	0.0 to 3276.7
money2	0.00 to 327.67
money3	0.000 to 32.767

Datatype	Range of values
money4	0.0000 to 3.2767
money5	0.00000 to 0.32767
money6	0.000000 to 0.032767
money7	0.0000000 to 0.0032767
money8	0.00000000 to 0.00032767
money9	0.000000000 to 0.000032767
money10	0.0000000000 to 0.0000032767
money11	0.00000000000 to 0.00000032767
money12	0.000000000000 to 0.000000032767
money13	0.0000000000000 to 0.0000000032767
money14	0.00000000000000 to 0.00000000032767
money15	0.000000000000000 to 0.000000000032767
bigdatetime	Current time with microseconds
date	Current time with seconds

Note: Values are not necessarily evenly distributed within these ranges.

Socket FIX Input Adapter

Adapter type: fixsocket_in. The Socket FIX Input adapter reads FIX messages from a TCP server socket and writes them as stream records.

Each stream hosts FIX messages of a certain type. The adapter discards messages of any other FIX type, and stores FIX fields in the same order in stream columns. The following fields are exceptions to this:

- BeginString
- BodyLength
- MsgType
- CheckSum

Ensure that the names of the stream columns correspond to the FIX protocol specification.

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is fixsocket_in.

Property Label	Property ID	Type	Description
FIX Version	fixVersion	choice	(Required) Version of the FIX protocol. Default value is 4.2.
FIX Message Type	fixMessageType	string	(Required) Type of messages hosted by the stream. No default value.
Source Host	fixHost	string	(Required) Name or IP address of source server for FIX messages. Default value is local-host.
Source Port	fixPort	uint	(Required) Port on which FIX messages are available on. Default value is 12345.
Reconnect Interval	reconnectInterval	uint	(Required) Reconnect interval, in seconds. If zero, makes no attempt to reconnect. Default value is 10.
Maximum Reconnect Attempts	maxReconnectAttempts	uint	(Required) Maximum number of reconnect attempts. Default value is zero.
Date Format	dateFormat	string	(Advanced) Date format. Default value is YYYY-MM-DDTHH:MM:SS.SSS.
Timestamp Format	timestampFormat	string	(Advanced) Timestamp format. Default value is YYYY-MM-DDTHH:MM:SS.SSS.
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.

Known limitations:

- This adapter is not a full FIX Engine.
- Supports only FIX versions 4.2, 4.3, 4.4, and 5.0.
- Does not support repeating groups and components.
- Supports only INSERT opcode.

See also

- *FIX Adapter* on page 135

Datatype Mapping for the Socket FIX Input Adapter

Event Stream Processor datatypes map to FIX datatypes.

Event Stream Processor Datatype	QuickFix Datatype
integer	boolean
string	byte[]
string	char
string	string
date	date
float	float
integer	integer
date or timestamp	UTCDateOnly
date or timestamp	UTCTimeOnly
date or timestamp	UTCTimeStamp

Socket FIX Output Adapter

Adapter type: `fixsocket_out`. The Socket FIX Output adapter writes stream data as FIX messages to a TCP server socket.

Each stream hosts FIX messages of a certain type. The adapter sends messages contiguously, with no line feeds. It generates the following FIX fields:

- BeginString
- BodyLength
- MsgType
- CheckSum

CHAPTER 2: Adapters Supported by Event Stream Processor

Ensure that the rest of the fields are stored in the appropriate order in stream columns, and that the names of the stream columns correspond to the FIX protocol specification.

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `fixsocket_out`.

Property Label	Parameter ID	Type	Description
FIX Version	fixVersion	<code>choice</code>	(Required) Version of the FIX protocol. Default value is 4.2.
FIX Message Type	fixMessageType	<code>string</code>	(Required) Type of messages hosted by the stream. No default value.
Destination Host	fixHost	<code>string</code>	(Required) Name or IP address of destination server for FIX messages. Default value is <code>localhost</code> .
Destination Port	fixPort	<code>uint</code>	(Required) Port on which the destination server socket is listening to FIX messages. Default value is 12346.
Date Format	dateFormat	<code>string</code>	(Advanced) Date format. Default value is <code>YYYY-MM-DDTHH:MM:SS.SSS</code> .
Timestamp Format	timestampFormat	<code>string</code>	(Advanced) Timestamp format. Default value is <code>YYYY-MM-DDTHH:MM:SS.SSS</code> .
PropertySet	propertyset	<code>string</code>	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.

Known limitations:

CHAPTER 2: Adapters Supported by Event Stream Processor

- This adapter is not a full FIX Engine.
- Supports only FIX versions 4.2, 4.3, 4.4, and 5.0.
- Does not support repeating groups and components.
- Does not attempt to reconnect if the connection to the FIX server is lost.
- Supports only INSERT opcode.

See also

- *FIX Adapter* on page 135

Datatype Mapping for the Socket FIX Output Adapter

Event Stream Processor datatypes map to FIX datatypes.

Event Stream Processor Datatype	QuickFix Datatype
integer	boolean
string	byte[]
string	char
string	string
date	date
float	float
integer	integer
date or timestamp	UTCDateOnly
date or timestamp	UTCTimeOnly
date or timestamp	UTCTimeStamp

Socket (As Client) CSV Input Adapter

Adapter type: `dsv_sockout_in`. The Socket (as Client) CSV Input adapter receives data in delimited format from outgoing network adapters.

The adapter initiates the connection to an external datasource, and an external program sends out the data. The data does not require a header (accepted by `esp_convert`). If the file includes a header, the header specifies the field names.

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `dsv_sockout_in`.

Sample record formats for the data file:

```
1. hasHeader=true  
delimiter=,
```

```
expectStreamNameOpcode=false
```

```
Ts,ItemID,Price,Quantity,WarehouseZipCode,DeliveryZipCode
2004/06/17 10:00:00.000000,SKU1276532,50.00,1,10012,94086
2004/06/17 10:00:05.000000,SKU6723143,23.00,2,10012,94043
```

```
2. expectStreamNameOpcode=true
delimiter=,
```

```
Trades_in,i,2004/06/17
10:00:00.000000,SKU1276532,50.00,1,10012,94086
Trades_in,i,2004/06/17
10:00:05.000000,SKU6723143,23.00,2,10012,94043
```

```
3. expectStreamNameOpcode=false
timestampFormat=%Y/%m/%d %H:%M:%S
delimiter=,
```

```
2004/06/17 10:00:00.000000,SKU1276532,50.00,1,10012,94086
2004/06/17 10:00:05.000000,SKU6723143,23.00,2,10012,94043
```

Property Label	Property ID	Type	Description
Server	host	string	(Required) Server host name. Default value is localhost.
Port	port	int	(Required) Server port. If port is set to -1, the adapter reads from the Ephemeral Port File. Default value is 12345.
Stream name, opcode expected	expectStreamNameOpcode	boolean	(Optional) If true, the adapter interprets the first two fields as a stream name and opcode respectively. Adapters discard messages with unmatched stream names. Default value is false.
Field Count	fieldCount	uint	(Optional) Counts the number of fields in a CSV file, if different from the source stream. Default value is 0.
Delimiter	delimiter	string	(Advanced) Symbol used to separate the columns. Default value is a comma (,).

Property Label	Property ID	Type	Description
Has Header	hasHeader	boolean	(Advanced) Determines whether the first line of the file contains the description of the fields. Default value is false.
Ephemeral Port File	epFile	file-name	(Advanced) File that contains the server port number, if Port is -1. No default value.
Retry Period	retryperiod	uint	(Advanced) Period for trying to re-establish an outgoing connection. In seconds. Default value is 1.
Enter Initial State	initial	choice	(Advanced) When the adapter enters the initial loading state. Default value is never.
Convert to Safe Opcodes	safeOps	boolean	(Advanced) Converts the opcodes INSERT and UPDATE to UPSERT. Converts DELETE to SAFEDELETE. Default value is false.
Skip Deletes	skipDels	boolean	(Advanced) Skips the rows with opcodes DELETE or SAFEDELETE. Default value is false.
Timestamp Format	timestampFormat	string	(Advanced) Format string for parsing timestamp values. Default value is %Y-%m-%dT%H:%M:%S.
Date Format	dateFormat	string	(Advanced) Format string for parsing date values. Default value is %Y-%m-%dT%H:%M:%S.
Block Size	blockSize	int	(Advanced) Determines number of records to block into one pseudo-transaction. Default value is 1.

Property Label	Property ID	Type	Description
Field Mapping	permutation	permutation	(Advanced) Mapping between the in-platform and external fields. No default value.
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.

Known limitations:

- The adapter ignores the stream name in the file rows.
- All data is sent to the same stream.

Socket (as Client) CSV Output Adapter

Adapter type: `dsv_sockout_out`. The Socket (as Client) CSV Output adapter sends data in delimited format to the outgoing network.

The Socket (as Client) CSV Output adapter initiates the connection to an external datasource and sends out the data. The data does not require a header (accepted by **esp_convert**). If the file includes a header, it specifies the field names. The adapter retries a connection if the connection breaks.

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `dsv_sockout_out`.

Property Label	Property ID	Type	Description
Server	host	string	(Required) Server host name. Default value is localhost.
Port	port	int	(Required) Server port. If port is set to -1, the adapter reads from the Ephemeral Port File. Default value is 12345.

CHAPTER 2: Adapters Supported by Event Stream Processor

Property Label	Property ID	Type	Description
Prepend stream name, opcode	prependStreamNameOpcode	boolean	(Optional) If true, the first two fields are interpreted as stream name and opcode respectively. The adapter discards messages with unmatched stream names. Default value is false.
Delimiter	delimiter	string	(Advanced) Symbol used to separate the columns. Default value is a comma (,).
Has Header	hasHeader	boolean	(Advanced) Determines whether the first line of the file contains the description of the fields. Default value is false.
Ephemeral Port File	epFile	file-name	(Advanced) File that contains the server port number, if port is -1. No default value.
Retry Period, s	retryperiod	uint	(Advanced) Period for trying to re-establish an outgoing connection, in seconds. Default value is 1.
Include Base Content	outputBase	boolean	(Optional) Starts by recording the initial contents of the stream, not just the updates. Default value is false.
Only Base Content	onlyBase	boolean	(Advanced) Sends the initial contents of the stream once. Default value is false.
Timestamp Format	timestampFormat	string	(Advanced) Format string for parsing timestamp values. Default value is %Y-%m-%dT%H:%M:%S.
Date Format	dateFormat	string	(Advanced) Format string for parsing date values. Default value is %Y-%m-%dT%H:%M:%S.
Field Mapping	permutation	permutation	(Advanced) Mapping between the in-platform and external fields. No default value.

Property Label	Property ID	Type	Description
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.

Socket (As Client) XML Input Adapter

Adapter type: `xml_sockout_in`. The Socket (As Client) XML Input adapter receives data in Event Stream Processor format from the outgoing network adapters.

The adapter initiates a connection with an outgoing network adapter, which can then send data to the input adapter. It is possible for the data not to have the header, or for the header not to specify the field names.

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `xml_sockout_in`.

Sample record format for the data file:

```
<Trades Id="0" Symbol="EBAY" TradeTime="2000-05-04T12:00:00"
Price="140.0" Shares="50" />
<Trades Id="1" Symbol="EBAY" TradeTime="2000-05-04T12:00:01"
Price="150.0" Shares="500" />
```

Property Label	Property ID	Type	Description
Server	host	string	(Required) The server host name. Default value is localhost.
Port	port	int	(Required) Server port. If port is set to -1, the adapter reads from the Ephemeral Port File. Default value is 12345.
Match Stream Name	matchStream-Name	boolean	(Optional) Ignores messages if the XML element name does not match the source stream name. Default value is false.
Ephemeral Port File	epFile	file-name	(Advanced) The file that contains the server port number, if port is set to -1. Default value is false.

Property Label	Property ID	Type	Description
Retry period (seconds)	retryperiod	uint	(Advanced) Indicates the time period for attempting to re-establish an outgoing connection, in seconds. Default value is 1.
Enter Initial State	initial	choice	(Advanced) Indicates when the adapter enters the initial loading state. Default value is never.
Convert to Safe Opcodes	safeOps	boolean	(Advanced) Converts the opcodes INSERT and UPDATE to UPSERT, and DELETE to SAFEDELETE. Default value is false.
Skip Deletes	skipDels	boolean	(Advanced) Skips the rows with opcodes DELETE or SAFEDELETE. Default value is false.
Date Format	dateFormat	string	(Advanced) Format string for parsing date values. Default value is %Y-%m-%dT%H:%M:%S.
Timestamp Format	timestampFormat	string	(Advanced) Format string for parsing timestamp values. Default value is %Y-%m-%dT%H:%M:%S.
Block Size	blockSize	int	(Advanced) Number of records to block into one pseudo-transaction. Default value is 1.
Field Mapping	permutation	permutation	(Advanced) Mapping between the in-platform and external fields. No default value.
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.

Known limitations:

- The adapter ignores the stream name in the file rows.

- All the data is sent to the same stream.

Socket (As Client) XML Output Adapter

Adapter type: `xml_sockout_out`. The Socket (As Client) XML Output adapter sends data in Event Stream Processor format to the outgoing network adapter.

The adapter initiates a connection with another program and then sends the data. If the connection is broken, the adapter retries the connection.

You can configure this adapter to send only the base state of the stream. The adapter sends data once and exits, but can be restarted later.

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `xml_sockout_out`.

Property Label	Property ID	Type	Description
Server	host	string	(Required) The server host name. Default value is localhost.
Port	port	int	(Required) Server port. If port is set to -1, the adapter reads from the Ephemeral Port File. Default value is 12345.
Include Base Content	outputBase	boolean	(Optional) Records the initial contents of the stream and not just the updates. Default value is false.
Ephemeral Port File	epFile	file-name	(Advanced) The file that contains the server port number, if port is -1. No default value.
Retry period, s	retryperiod	uint	(Advanced) The time period for attempting to re-establish an outgoing connection, in seconds. Default value is 1.
Only Base Content	onlyBase	boolean	(Advanced) Sends only the initial contents of the stream. Default value is false.
Date Format	dateFormat	string	(Advanced) The format string for parsing date values. Default value is %Y-%m-%dT%H:%M:%S.
Timestamp Format	timestampFormat	string	(Advanced) The format string for parsing timestamp values. Default value is %Y-%m-%dT%H:%M:%S.

Property Label	Property ID	Type	Description
Field Mapping	permutation	permutation	(Advanced) Mapping between the in-platform and external fields. Default value is %Y-%m-%dT%H:%M:%S.
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.

Socket (As Server) XML Input Adapter

Adapter type: `xml_sockin_in`. The Socket (As Server) XML Input adapter receives data in Event Stream Processor format from the incoming network adapter.

Another program initiates the connection and then sends the data.

This adapter can be configured to send only the base state of the stream, and can be repeatedly reconnected.

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `xml_sockin_in`.

Property Label	Property ID	Type	Description
Port	port	int	(Required) Server port. If port is set to -1, the adapter reads from the Ephemeral Port File. Default value is 12345.
Match Stream Name	matchStreamName	boolean	(Optional) If true, the XML element names are matched against the stream name. Unmatched messages are discarded. Default value is false.
Ephemeral Port File	epFile	filename	(Advanced) The file that contains the port number, if port is -1. No default value.
Initial Listen Period (seconds)	retryperiod	uint	(Advanced) Designates the length of time to wait for the first incoming connection before switching to the continuous state. Default value is 0.

Property Label	Property ID	Type	Description
Enter Initial State	initial	choice	(Advanced) Indicates when the adapter enters the initial loading state. Default value is never.
Convert to Safe Opcodes	safeOps	boolean	(Advanced) Converts the opcodes INSERT and UPDATE to UPSERT, and DELETE to SAFEDELETE. Default value is false.
Skip Deletes	skipDels	boolean	(Advanced) Skips the rows with opcodes DELETE or SAFEDELETE. Default value is false.
Date Format	dateFormat	string	(Advanced) The format string for parsing date values. Default value is %Y-%m-%dT%H:%M:%S.
Timestamp Format	timestampFormat	string	(Advanced) The format string for parsing timestamp values. Default value is %Y-%m-%dT%H:%M:%S.
Block Size	blockSize	int	(Advanced) Number of records to block into one pseudo-transaction. Default value is 1.
Field Mapping	permutation	permutation	(Advanced) Mapping between the in-platform and external fields. No default value.
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.

Known limitations:

- The adapter ignores the stream name in the file entries.
- All the data is sent to the same stream.
- Supports only one network connection at a time.

Socket (As Server) XML Output Adapter

Adapter type: `xml_sockin_out`. The Socket (As Server) XML Output adapter receives data in Event Stream Processor format from the outgoing network adapters.

Another program initiates the connection and then receives the data from the output adapter.

This adapter can be configured to send only the base state of the stream. The socket closes after sending the base state of the stream but can be repeatedly reconnected.

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `xml_sockin_out`.

Property Label	Property ID	Type	Description
Port	port	int	(Required) Server port. If port is set to -1, the adapter reads from the Ephemeral Port File. Default value is 12345.
Include Base Content	outputBase	boolean	(Optional) Starts by recording the initial contents of the stream, not just the updates. Default value is false.
Ephemeral Port File	epFile	filename	(Advanced) The file that contains the port number, if port is -1. No default value.
Only Base Content	onlyBase	boolean	(Advanced) The adapter sends the initial contents of the stream, once. Default value is false.
Date Format	dateFormat	string	(Advanced) The format string for parsing date values. Default value is %Y-%m-%dT%H:%M:%S.
Timestamp Format	timestampFormat	string	(Advanced) The format string for parsing timestamp values. Default value is %Y-%m-%dT%H:%M:%S.
Field Mapping	permutation	permutation	(Advanced) Mapping between the in-platform and external fields. No default value.

Property Label	Property ID	Type	Description
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.

Known limitations:

- Supports only one network connection at a time.

Socket (As Server) CSV Input Adapter

Adapter type: `dsv_sockin_in`. The Socket (As Server) CSV Input adapter receives data in Event Stream Processor delimited format from the incoming network adapters.

Another program initiates the connection and then sends the data to the adapter.

It is possible for the data not to have the header, or for the header not to specify the field names.

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `dsv_sockin_in`.

Property Label	Property ID	Type	Description
Port	port	int	(Required) Server port. If port is set to -1, the adapter reads from the Ephemeral Port File. Default value is 12345.
Stream name, opcode expected	expectStreamNameOpcode	boolean	(Optional) If true, the first two fields are interpreted as stream name and opcode respectively. Messages with unmatched stream names are discarded. Default value is false.
Delimiter	delimiter	string	(Advanced) Symbol used to separate the columns. Default value is a comma (,).
Has Header	hasHeader	boolean	(Advanced) Determines whether the first line of the file contains the description of the fields. Default value is false.

Property Label	Property ID	Type	Description
Ephemeral Port File	epFile	file-name	(Advanced) The file that contains the server port number, if port is -1.
Initial Listen Period (seconds)	retryperiod	uint	(Advanced) How long to wait for the first incoming connection before switching to the continuous state. Default value is 0.
Enter Initial State	initial	choice	(Advanced) Indicates when the adapter enters the initial loading state. Default value is never.
Convert to Safe Opcodes	safeOps	boolean	(Advanced) Converts the opcodes INSERT and UPDATE to UPSERT, and DELETE to SAFEDELETE. Default value is False.
Skip Deletes	skipDels	boolean	(Advanced) Skips the rows with opcodes DELETE or SAFEDELETE. Default value is false.
Date Format	dateFormat	string	(Advanced) The format string for parsing date values. Default value is %Y-%m-%dT%H:%M:%S.
Timestamp Format	timestampFormat	string	(Advanced) The format string for parsing timestamp values. Default value is %Y-%m-%dT%H:%M:%S.
Block Size	blockSize	int	(Advanced) The number of records to block into one pseudo-transaction. Default value is 1.
Field Mapping	permutation	permutation	(Advanced) Mapping between the in-platform and external fields. No default value.
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.

Known limitations:

- The stream name in the file rows is ignored.
- All the data is sent to the same stream.
- Supports only one network connection.

Socket (As Server) CSV Output Adapter

Adapter type: `dsv_sockin_out`. The Socket (As Server) CSV Output adapter sends data in Event Stream Processor delimited format to the incoming network adapters.

The adapter can be configured to send only the base state of the stream. The socket closes after sending the base state of the stream but may be repeatedly reconnected.

It is possible for the data not to have the header, or for the header not to specify the field names.

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `dsv_sockin_out`.

Property Label	Property ID	Type	Description
Port	port	<code>int</code>	(Required) Server port. If port is set to -1, the adapter reads from the Ephemeral Port File. Default value is 12345.
Include Base Content	outputBase	<code>boolean</code>	(Optional) Records the initial contents of the stream, not just the updates. Default value is false.
Prepend Stream Name, Opcode	prependStream-NameOpcode	<code>boolean</code>	(Optional) If true, each message starts with the stream name and the opcode. Default value is false.
Ephemeral Port File	epFile	<code>filename</code>	(Advanced) The file that contains the server port number, if port is -1. No default value.
Only Base Content	onlyBase	<code>boolean</code>	(Advanced) The adapter sends only the initial contents of the stream, once. Default value is false.
Delimiter	delimiter	<code>string</code>	(Advanced) The symbol used to separate the columns. Default value is a comma (,).
Has Header	hasHeader	<code>boolean</code>	(Advanced) Whether the first line of the file contains the description of the fields. Default value is false.

Property Label	Property ID	Type	Description
Date Format	dateFormat	string	(Advanced) The format string for parsing date values. Default value is %Y-%m-%dT%H:%M:%S.
Timestamp Format	timestampFormat	string	(Advanced) The format string for parsing timestamp values. Default value is %Y-%m-%dT%H:%M:%S.
Field Mapping	permutation	permutation	(Advanced) Mapping between the in-platform and external fields. No default value.
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.

Known limitations:

- Supports only one network connection.

SMTP Output Adapter

Adapter type: `smtp_out`. The SMTP Output adapter sends an e-mail containing stream records.

For each record, the e-mail body contains:

- Stream name
- Columns names and values

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `smtp_out`.

Property Label	Property ID	Type	Description
SMTP Host	smtpHost	string	(Required) Name or IP address of the e-mail server. No default value.

Property Label	Property ID	Type	Description
fromAddress	from	string	(Required) E-mail address of the sender. No default value.
Importance Column	importanceColumn	string	(Required) Name of the stream column where the e-mail importance is stored. Valid values are: high, normal, and low. The default value is importance. The values are case-sensitive.
Address Column	addressColumn	string	(Required) Name of the column where a semicolon-delimited list of recipient e-mail addresses is stored. No default value.
Subject Column	subjectColumn	string	(Required) Name of the stream column where the e-mail subject is stored. No default value.
SMTP Port	smtpPort	uint	(Optional) Port used by the SMTP server. Default value is 25.
SMTP Username	smtpUsername	string	(Optional) Once you configure this property, the SMTP authentication requires the SMTP password to be set also.
SMTP Password	smtpPassword	string	(Optional) Set this property if you have set a SMTP username.

Property Label	Property ID	Type	Description
Use SSL	useSSL	boolean	<p>(Optional) If you want to enable this option, import the security certificate into the Java keystore file (located under <code>install/lib/jre/lib/security/cacerts</code>). Once the security certificate is configured, you can enable this option.</p> <p>Default value is false.</p> <hr/> <p>Important: Enable the SMTP server for SSL before configuring this option.</p>
Use TLS	useTLS	boolean	<p>(Optional) If you want to enable this option, import the security certificate into the Java keystore file (located under <code>install/lib/jre/lib/security/cacerts</code>). Once the security certificate is configured, you can enable this option.</p> <p>Default value is false.</p> <hr/> <p>Important: Enable the SMTP server for TLS before configuring this option.</p>

Property Label	Property ID	Type	Description
cc Column	ccColumn	string	(Advanced) Name of the column where a semicolon-delimited list of recipient cc addresses is stored. By default, no cc e-mails are sent.
bcc Column	bccColumn	string	(Advanced) Name of the column where a semicolon-delimited list of recipient bcc addresses is stored. By default, no bcc e-mails are sent.
Column Names	columnNames	string	(Advanced) Colon-delimited names of stream columns whose values are included in the e-mail. By default, the e-mail contains values of all columns in the stream.
Show Column Names	showColumnNames	boolean	(Advanced) If true, the adapter includes the column names in the e-mail along with their values. If false, it includes only the values. Default value is true.

Property Label	Property ID	Type	Description
Number of Resend Attempts	resendAttempts	integer	<p>(Advanced) The number of times to retry sending an e-mail if the initial attempt to send it fails. Default is 0, no attempt is made to resend e-mails.</p> <p>Choose a moderate value (0 - 10) for this property. Requiring a large number of attempts to resend the e-mail may lead to excessive memory consumption, particularly if aggravated by network problems and a high volume of records waiting to be e-mailed.</p>
Log Alert	logAlert	boolean	<p>(Advanced) If true, logs an alert at debug level 1 each time the e-mail sending has been successful or failed. Default value is true.</p>
Date Format	dateFormat	string	<p>(Advanced) Date format. Default value is %Y-%m-%dT%H:%M:%S.</p>
Timestamp Format	timestampFormat	string	<p>(Advanced) Timestamp format. Default value is %Y-%m-%dT%H:%M:%S.</p>

Property Label	Property ID	Type	Description
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.

Known limitations:

- If you are a Microsoft Outlook® user, disable the feature that removes extra line breaks:
 1. Open Outlook, go to **Tools > Options**.
 2. On the **Preferences** tab, select **E-mail Options**.
 3. Click to clear the **Remove extra line breaks in plain text messages** check box. Click **OK** twice.

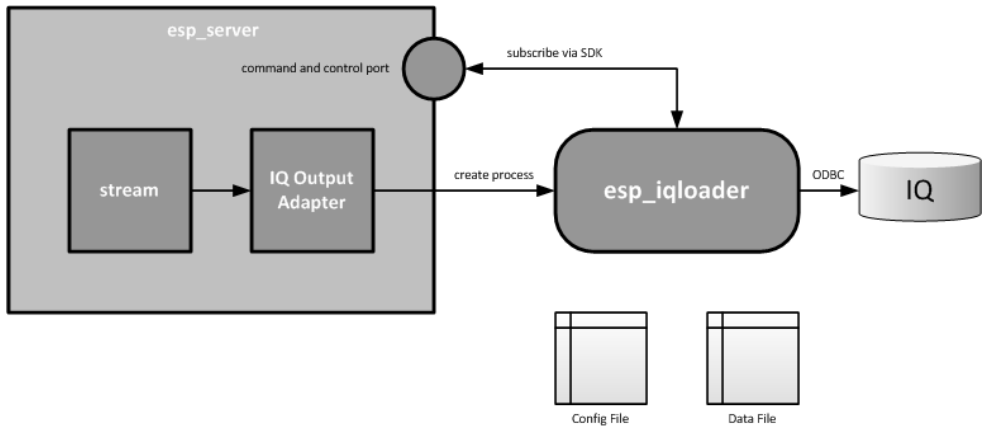
Sybase IQ Output Adapter

Adapter type: `sybase_iq_out`. The Sybase IQ Output adapter loads data from Event Stream Processor into Sybase IQ.

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `sybase_iq_out`.

The Sybase IQ Output adapter makes use of the **esp_iqloader** utility to load data into Sybase IQ. When the adapter starts, it launches the **esp_iqloader** utility, which subscribes to the streams specified in its configuration file, and loads the data from those streams into IQ using ODBC.

CHAPTER 2: Adapters Supported by Event Stream Processor



Property Label	Property ID	Type	Description
Configuration File	configFile	filename	(Required) Full path to the esp_iqloader configuration file that provides Sybase IQ connection details, stream information, and load options. See the <i>Utilities Guide</i> for more information about the format of this file.
User Name	user	string	(Optional) User name the esp_iqloader utility uses when connecting to Event Stream Processor. Note: This property is required when using Server RSA, username or password, Kerberos, and LDAP authentication.
Password	password	password	(Optional) Password to use to connect to the Event Stream Processor. Note: Not needed if specifying the RSA Key file.

Property Label	Property ID	Type	Description
RSA Key File	rsaKeyFile	RuntimeFilename	<p>(Optional) Full path to the RSA key private key file. Set this when using Server RSA authentication to the Event Stream Processor.</p> <hr/> <p>Note: Set a user name before configuring this option.</p>
Use Kerberos	useKerberos	boolean	<p>(Optional) If true, uses Kerberos authentication to the Event Stream Processor if the RSA key file parameter is not specified. If the RSA key file parameter is specified and UseKerberos is set to true, the RSA settings override Kerberos settings and RSA authentication is used. Default value is false.</p> <hr/> <p>Note: Set a user name before configuring this option.</p>
PropertySet	propertyset	string	<p>(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.</p>

Property Label	Property ID	Type	Description
Archive Deltas	archiveDeltas	boolean	<p>(Advanced) If true, archives deltas, otherwise archives only a snapshot of the data.</p> <p>Delta mode continually uploads all data (existing and updates) from the specified streams or windows into Sybase IQ. Snapshot mode uploads all existing data from the specified streams or windows when the adapter starts up, and then exits.</p> <p>Existing data is data that is already present in the window when the adapter starts up and connects to it. Default value is true.</p>
Project URI	projectUri	string	<p>(Required) The URI needed to connect to a project in the Event Stream Processor cluster. For example, <host-name>:<port>/<work-space-name>/<project-name>.</p>
Swap Bytes	byteSwap	boolean	<p>(Advanced) Set to true if the Event Stream Processor and the esp_iqloader utility are running on different architectures (little/big endian). Default value is false.</p>
Recover Only	recoverOnly	boolean	<p>(Advanced) If true, recovers any data that was read from the Event Stream Processor but not archived, and exits. Default value is false.</p>
Datawarehousing Mode	dataWarehouse	boolean	<p>(Advanced) If true, updates are treated as inserts and deletes are ignored. Default value is false.</p>

Property Label	Property ID	Type	Description
Archive Interval	archiveInterval	uint	(Advanced) Specifies how long to wait, in seconds, after each time a set of data is archived and the next set of data is archived. Default value is 1.
Precision	precision	uint	(Advanced) A value between 0 and 6 that specifies the precision to use for money and float data-types. Default value is 6.
Subscription Buffer Size	queueSize	uint	(Advanced) A positive value greater than 1000 that specifies the subscription buffer size to use. Note: Change this value if the data is bursty and the Event Stream Processor reports the subscription buffer is full.
Commit Batch Size	batchSize	uint	(Advanced) The batch size used to archive data into Sybase IQ. Default value is 1000. Important: Specify bulk loads in the batch size configuration file.
ODBC Retry Attempts	odbcRetryTimes	uint	(Advanced) Number of times to retry the ODBC connection if a connection cannot be made. Default value is 5.
ODBC Retry Interval	odbcRetryInterval	uint	(Advanced) Number of seconds to wait before retrying the ODBC connection. Default value is 60.

Known limitations:

- Attaching a Sybase IQ output adapter to a stream does not guarantee that the adapter archives the data from this stream. The adapter uses the specified configuration file to get this information. If a different stream is specified by the configuration file, then that one is archived.

Datatype Mapping for the Sybase IQ Adapter

Event Stream Processor datatypes map to Sybase IQ datatypes.

If required, you can create a table in Sybase IQ to map Event Stream Processor datatypes to other compatible Sybase IQ datatypes. The datatype mapping below is default mapping.

Note: The date and timestamp Event Stream Processor datatypes are compatible only with the `datetime` Sybase IQ datatype.

Event Stream Processor Datatypes	Sybase IQ Datatypes
<code>bigdatetime</code>	<code>timestamp</code>
<code>binary</code>	<code>binary</code>
<code>boolean</code>	<code>varchar(5)</code>
<code>date</code>	<code>datetime</code>
<code>float</code>	<code>float</code>
<code>integer</code>	<code>integer</code>
<code>interval</code>	<code>bigint</code>
<code>long</code>	<code>bigint</code>
<code>money</code>	<code>decimal(38,4)</code>
<code>money(n)</code>	<code>decimal(38,n)</code>
<code>string</code>	<code>varchar(n)</code>
<code>timestamp</code>	<code>datetime</code>

WebSphere MQ Adapter

Event Stream Processor supplies WebSphere MQ adapters that enable you to read and write to and from the WebSphere MQ queue and an Event Stream Processor stream.

Event Stream Processor permits a WebSphere MQ server to read and write to the Event Stream Processor engine. You can customize these internal adapters to suit your needs. Because WebSphere MQ messages are unstructured, properly define a schema and prepare the MQ messages. The full range of Event Stream Processor datatypes is permitted in the schema definition. You can send binary data, strings, and so on, into and out of the Event Stream Processor engine.

Ensure MQ Client adapters have MQ 7.0.1 Client software installed. Failure to match the adapter to installed software results in errors. Sybase WebSphere MQ adapters are designed to

work with WebSphere MQ client software on the same host computer as the Server. The WebSphere MQ Server, however, can reside on the same computer or on another computer.

WebSphere MQ Input and Output adapters support opcodes for inserting, deleting, updating, and upserting data between the Event Stream Processor and WebSphere queues.

WebSphere MQ Input Adapter

Adapter type: `wsmq_in`. The default WebSphere MQ Input adapter reads a string in CSV format.

Ensure the order of the data in the message matches the schema of the input stream to which the adapter is attached. The WebSphere MQ Input adapter applies stream names and opcode instructions (INSERT, DELETE, UPDATE, UPSERT) to CSV data pulled from a queue. See the **expectStreamNameOpcode** property.

Note: If you are reading data from WSMQ using the WebSphere MQ Input adapter, but publishing data using a JMS program rather than the WebSphere MQ Output adapter, set the following line in the JMS program:

```
queue.setTargetClient(com.ibm.mq.jms.JMSC.MQJMS_CLIENT_NONJMS_MQ);
```

To run the adapter successfully in Linux and UNIX installations, set the `LD_LIBRARY_PATH` to point to the MQ client libraries.

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `wsmq_in`.

Note: To avoid data loss in the case of a server failure, Sybase recommends you attach a log store to an input window.

Property Label	Property ID	Type	Description
Queue Name	QueueName	string	(Required) The name of the queue on the server to send messages. This queue must be managed by the indicated Queue Manager Name. No default value.
Queue Manager Name	QueueManagerName	string	(Required) The name of the queue manager on the server to send messages to. No default value.
MQ System Name	SystemName	string	(Required) The name of the MQ server system. This may be a symbolic name or an IP address. No default value.

Property Label	Property ID	Type	Description
Port	Port	string	(Required) The port number on the MQ server system to which the MQ server queue listener is attached. No default value.
MQ Channel	Channel	string	(Required) The name of the MQ server channel associated with the queue. No default value.
Maximum Input Buffer Size	MaxBufferSize	uint	(Required) The maximum size of the buffer, in bytes. No default value.
Sync Point Commit Mode	syncpointmode	boolean	(Optional) Enables sync point commit mode. Set this to ensure guaranteed delivery of messages from the adapter to the Server. Default value is false.
Batch Size	batchsize	uint	(Optional) Specifies number of records in a batch to commit in sync point commit mode. Default value is 1.
CSV Delimiters	CsvSeparators	string	(Optional) The CSV field separators. Can be multiple characters. Default value is a comma (,).
CSV Escape Characters	CsvEscapeChars	string	(Optional) The character that escapes the meaning of special characters, including the delimiters, escape characters, and quote characters. Can be multiple characters. Default value is a backslash (\).
CSV Quote Characters	CsvQuoteChars	string	(Optional) The characters to delineate the beginning and end of a field. Default value is double quotes (").
Perform CSV Trimming	CsvTrimming	boolean	(Optional) If set to true, this trims leading and trailing whitespace from the beginning and end of each field. If true, a quoted field containing nothing but spaces is interpreted as NULL. Default value is true.

Property Label	Property ID	Type	Description
Stream Name, Opcode Expected	TimestampColumnFormat	string	(Optional) The format for timestamp values. Default value is YYYY-MM-DDTHH:MM:SS.SSS.
Timestamp Column Format	expectStreamNameOpcode	boolean	(Advanced) If true, the first two fields in CSV records are interpreted as stream name, opcode. Default value is false.
Date Column Name	DateColumnName	string	(Advanced) The format in which date values are stored in the file. Default value is YYYY-MM-DDTHH:MM:SS.SSS.
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.

See also

- *Chapter 5, Guaranteed Delivery* on page 543

WebSphere MQ Output Adapter

Adapter type: wsmq_out. The default WebSphere MQ Output adapter publishes a string in CSV format.

The WebSphere MQ adapter does not produce a header line because the schema of the stream publishing to the adapter determines the order and datatypes of the fields. Columns are published in the default display format for the appropriate datatype. The adapter prepends stream names and opcode instructions (insert, delete, update, upsert) to CSV data added to a queue. See the **prependStreamNameOpcode** property.

To run the adapter successfully in Linux and UNIX installations, set the `LD_LIBRARY_PATH` to point to the MQ client libraries to run the adapter successfully.

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `wsmq_out`.

Note: This adapter uses TCP/IP for transfers. To use other protocols, determine the appropriate configuration and interface properties for those protocols.

Property Label	Property ID	Type	Description
Queue Name	QueueName	string	(Required) Name of the queue on the server to send messages. This queue must be managed by the indicated Queue Manager Name. No default value.
Queue Manager Name	QueueManagerName	string	(Required) Name of the queue manager on the server to send messages. No default value.
MQ System Name	SystemName	string	(Required) Name of the MQ server system. This may be a symbolic name or an IP address. No default value.
Port	Port	string	(Required) Port number on the MQ server system to which the MQ server queue listener is attached. No default value.
MQ Channel	Channel	string	(Required) Name of the MQ server channel associated with the queue. No default value.
CSV Field Separator	CsvSeparatorChar	string	(Optional) The CSV field separator, specify a single character. Default value is a comma (,).
CSV Escape Character	CsvEscapeChar	string	(Optional) The character to escape the meaning of special characters, including the field separator, escape character, and quote character. Default value is a backslash (\).
CSV Quote Character	CsvQuoteChar	string	(Optional) The character to delineate the beginning and end of a field, which can include anything. Any embedded quote characters are escaped. Default value is a double quote (").
Prepend Stream Name, Opcode	prependStreamNameOpcode	boolean	(Advanced) If true, every CSV record is prepended with stream name and an opcode. Default value is false.
Timestamp Column Format	TimestampColumnFormat	string	(Advanced) The format for timestamp values. Default value is YYYY-MM-DDTHH:MM:SS.SSS.

Property Label	Property ID	Type	Description
Date Column Format	DateColumn-Format	string	(Advanced) The format in which date values are stored in the file. Default value is YYYY-MM-DDTHH:MM:SS.SSS.
Runs Adapter in GD Mode	enable-GDMode	boolean	(Advanced) If set to true, the adapter runs in guaranteed delivery (GD) mode and all GD-related parameters become required. Default value is false.
Name of Column Holding GD Key	gdKeyColumn	string	(Advanced) Specifies column name in the Flex operator holding the GD key. The GD key is a constantly increasing value that uniquely identifies every event regardless of the opcode in the stream of interest. No default value.
Name of Column Holding opcode	gdOpcodeColumn	string	(Advanced) Specifies name of column in Flex operator holding opcode. The opcode is the operation code (for example, inserts, update, or delete) of the event occurring in the stream of interest. No default value.
Name of Truncate Stream	gdControl-Stream	string	(Advanced) Specifies name of the control window in the GD setup. The control window is a source stream that informs the Flex operator of which data has been processed by the adapter and can be safely deleted. No default value.
Purge After Number of Records	gdPurgeInternal	int	(Advanced) Specifies number of records after which to purge the Flex operator. Default value is 1000.
Batch Size to Update Truncate Stream	gdBatchSize	int	(Advanced) Specifies number of records after which the control window must be updated with the latest GD key. Default value is 1000.

Property Label	Property ID	Type	Description
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.

See also

- *Chapter 5, Guaranteed Delivery* on page 543

Queue Configuration

Use this sample code to call a queue manager, server queue, channel, and listener.

A standard MQ server configuration provides:

- A default queue manager called `queue.manager.1`.
- A local server queue called `QUEUE1`.
- A Server-Connection channel called `channel1`.
- A listener called `listener1` on TCP/IP port 2001.

```

Create a default queue manager called queue.manager.1 and
start it:
crtmqm -q queue.manager.1
strmqm
dspmq # display list of active queues
Now create a local queue, a channel and a listener:
runmqsc
define qlocal(QUEUE1)
5
define channel (channel1) chltype (svrconn) trdtype (tcp)
\
mcauser ('mqm')
define listener (listener1) trdtype (tcp) control (qmgr) \
port (2001)
start listener (listener1)
end

```

Note: In this configuration example, backslashes (\) are used for readability, and because of space constraints. When configuring queues in the system, keep this information on one line.

External Adapters

Event Stream Processor provides specialized external adapters for processing nonstandard data formats from non-standard interfaces.

External adapters execute as a separate process either on the same machine as the Server or on different machines. These adapters start and stop independently of the Server and associated query modules.

See also

- *Adapter Parameters Datatypes* on page 8
- *Adapters that Support Schema Discovery* on page 539
- *Custom External Adapters* on page 517

ESP Add-In for Microsoft Excel

The Sybase ESP Add-in for Microsoft Excel® is a real-time data add-in for Microsoft Excel that lets you view and retrieve records from one or more running Event Stream Processor projects, as well as publish records to them.

The ESP Add-in for Microsoft Excel does not support Linux or Solaris platforms. You can run it on 32-bit and 64-bit editions of Windows, and with MS Excel 2007 and 2010 (32-bit editions only).

On the display side, you can use the ESP Add-in for Microsoft Excel to select streams and view the columns within those streams. You can also filter records based on data values, and view the most recent “N” records, or the most recent “N” records that match a specified filter (where “N” is the specified number of records).

On the publication side, you can use the ESP Add-in for Microsoft Excel to automatically publish data whenever data changes in a range of cells. You can also manually publish data to Event Stream Processor by selecting a range of cells and using the Publication wizard.

Connection Wizard

The Connection wizard enables you to simultaneously connect to one or more instances of Event Stream Processor.

The connection information for an Excel workbook is saved with the workbook.

Component	Description
Connections	Enter the name of a new connection, or select from a list of previously defined connections. When you select a connection, all the information associated with that connection appears.

Component	Description
Host	(Required) Enter the host name for the Event Stream Processor cluster manager to connect to.
Port	(Required) Enter the port of the Event Stream Processor cluster manager.
Workspace	Enter the workspace that the project is part of.
Project	Enter the project to connect to.
User	Enter the user name to connect to the cluster manager. This property is required unless you selected none as the authentication type.
Password	(Optional) Enter the password associated with the user name.
SSL	To connect to an Event Stream Processor cluster manager that was started with SSL mode enabled.
Authentication Type	Select an authentication type from the list. The authentication type selected must match the mode that was used when starting Event Stream Processor.
RSA Key File	If RSA authentication is selected, enter the RSA key file. You can either type the location and name of the key file, or click the button next to the field to browse and to choose the file.
Save	Saves the connection information in a hidden worksheet associated with the active workbook. Connection when the active workbook is retrieved and shown.
Connect	After providing all the information, click this button to connect to the server. If the connection is successful, only the Disconnect button is available for this connection. This also saves the connection information for future use.
Disconnect	Drop the connection to Event Stream Processor. On a successful disconnect, this button is disabled and the Connect and Delete buttons are enabled. Any queries that are actively using this connection are stopped, after user confirmation, before disconnecting.
Delete	Delete a connection.
Hide	Hide the window while preserving all information. To redisplay the screen, click the SybaseRT button on the Excel toolbar.

Subscription Wizard

The Subscription wizard pane enables you to define and control one or more subscription queries, the results of which appear in the Excel worksheet.

The ESP Add-in for Microsoft Excel keeps track of the locations of your queries even if its defined cells are shifted horizontally or vertically.

Component	Description
Subscription Queries	Enter the name of a new query, or choose a previously defined query. When you select a previously saved query, you see all information associated with that query.
Connection Name	Select the connection associated with the query. you can run a query only if the associated connection is active.
Start Cell	Enter the location in the Excel worksheet in which start inserting the real-time data formulas. Specify the location in “A1” notation; for example, a value of B5 tells the ESP Add-in for Microsoft Excel to insert the formulas as a grid starting at column B, row 5.
Max Rows	Enter the number of records (maximum value 65536) to appear in the Excel worksheet. When there are more rows to be displayed than the number specified, the oldest records are discarded.
Get Base Transactions	Select this field to retrieve base records from a stream before new transactions. Leave this field unselected to retrieve only new transactions. For small tables with relatively few new transactions turn this option on, so you can see query data. Otherwise, the data for the query is not displayed. However, for dynamic tables with high transaction activity, leave this option unselected: otherwise, Excel tries to potentially load millions of records every time it starts.
Streams	Displays all the streams available in the server with which the selected connection is associated. These streams automatically appear when you select a connection.
Columns	When one of the streams is selected, this area displays each column, along with its datatype and a check box to indicate whether or not it is a key column. You can choose a different key column for the stream than the specified one.

Component	Description
SQL Statement	To customize data retrieved from Event Stream Processor, specify a SQL statement. The statement cannot include joins, group by, and order by clauses as the SQL is applied to individual transaction logs for the stream, not the data in the stream. See the <i>Utilities Guide</i> for information about esp_query supported SQL syntax. This option is available only if SQL Statement is selected as the stream with which the connection is associated.
Parse SQL	Parse the SQL statement in the SQL Statement field. If the SQL is parsed successfully, the column names and corresponding data types appear in the Columns field. By default, while none of the columns are marked as key fields, the appropriate key columns must be selected before the real-time data query is applied.
Apply	Apply the real-time data formulas in the Excel worksheet after configuring a new subscription or modifying an existing subscription. Once the formulas have been applied, you can start the query.
Reset	Display the properties of the Subscription Query when it was last saved. Select this option if changes have been made to the query that need to be completely reversed.
Delete	Delete a previously saved query.
Start	Start the query. The data appears in the Excel worksheet.
Stop	Stop the running query. No new data appears in the Excel worksheet. However, any data that appears continues to show until you close and reopen the worksheet, or restart the query.

Publication Wizard

The Publication wizard lets you manually publish data to a stream and graphically construct publication formulas meant for automatic publishing.

Components	Description
Connection Name	The name of the connection to use for publishing. Only active connections appear. When you click a connection, the streams the connection object is connected to appear in the Streams field, and the columns and the datatypes for the stream appear in the table named Columns.
Operation Code	If a record exists, select UPDATE, DELETE, or UPSERT (the default). Otherwise, select INSERT.

Components	Description
Data Range	<p>Specify the range of cells in the Excel worksheet that contain the data to publish. You cannot edit this field directly: select the cells in the worksheet to publish, then click the blue button next to this field to populate it.</p> <p>You cannot simultaneously publish multiple noncontiguous areas.</p>
WorkBook Name	<p>A read-only field that shows the workbook in which the selected range is located.</p>
WorkSheet Name	<p>A read-only field that shows the worksheet in which the selected range is located.</p>
First Row Has Columns	<p>Indicate that the first row in the selected range has column names. If it does not, leave this field unselected.</p> <p>When you specify data columns, they can be in any order, and only values for the desired fields must be supplied. The rest of the columns are automatically filled with NULL.</p> <p>However, you do not provide column names, the ESP Add-in for Microsoft Excel expects all the columns in the streams to be in the exact same order as defined in Event Stream Processor.</p>
Transpose Rows To Columns	<p>Select this option if the data columns for a record are provided vertically in a single column instead of the horizontally across multiple columns (the normal way of representing records).</p>
Streams	<p>Select the stream for which a publication should be made.</p>
Columns	<p>This table shows the columns and the corresponding datatypes for the selected stream. You cannot select the columns in this table; however, you can copy the names of the columns and paste them in Excel.</p>
Log File	<p>(Optional) Specify the path and file name to which publication errors are written, either by entering it directly into the field, or by browsing to and selecting it.</p> <p>The errors written to this file in also appear in the Result field.</p>
Result	<p>Read-only results of the publication.</p>

Components	Description
Publish Data	<p>Publish the data to Event Stream Processor. The result of the publication appears in the Result field.</p> <p>The Event Stream Processor only acknowledges whether the data has been received. To find out whether the record was rejected for some reason, such as a duplicate insert or bad data, either subscribe to the stream or submit a SQL query.</p>
Show Formula	<p>Graphically create the formula. This provides a convenient way to create a formula for automatic publishing. If there are no errors, the formula appears in the Results field. You can then copy the formula and place it in the Excel worksheet to start automatically publishing the data.</p>
Clear Results	<p>Clear the Result field if there are too many entries.</p>

Automatic Publishing

Use the **SybaseRTP** add-in function when the data in a cell changes

SybaseRTP is a wrapper function around the underlying Excel real-time data mechanism used for publishing data. The syntax for this formula is:

```
=SybaseRTP("ConnectionName", "StreamName", "OperationCode", DataRange,
[[ColumnRange], [TransposeRows], ["LogFile"], [InstanceNo],
[NoResults]])
```

where:

Parameter	Description
ConnectionName	Name of the connection to use for publishing. You must establish the connection before you can publish successfully.
StreamName	Name of the stream where to publish data.
OperationCode	The opcode for publishing INSERT, UPDATE, DELETE, or UPSERT.
DataRange	The address or name of the Excel range containing the data to publish. Do not enclose the DataRange object in double quotes.
[ColumnRange]	(Optional) The Excel range address or range name containing the stream column names. Do not enclose this parameter in quotes.
[TransposeRows]	(Optional) Whether the data record is specified in a column instead of a row. It can be either true or false (the default).

Parameter	Description
[LogFile]	(Optional) The name and location of the log file to which any errors are logged. If not provided, no logging is done.
[InstanceNo]	Internal use only; leave empty.
[NoResults]	Internal use only; set to false or leave empty.

For example:

```
=SybaseRTP("Connection1","Trades","INSERT",A2:E10,A1:E1,False,"C:\logs\log1.log",,)
```

You can place this formula in any sheet in the workbook. When constructing the formula, tell Excel the workbook or worksheet to which the address refers either by selecting the appropriate cells in the desired worksheet or using the [Workbook]Worksheet!A1:E5 format.

Once the formula is in Excel any changes made to any of the cells publishes the entire range. To publish only when certain cells are changed, use a call inside a custom wrapper that encapsulates the business logic that dictates when to call this function.

The return value for this function is an array that is formatted as a string using an Excel-style location: {{val11,val12},{val21,22}....}. You can then convert this formula into an Excel-style array object. The string contains one or more array of elements, and each subelement contains two subitems. The array string contains only one element when there are errors passed in values. Otherwise, it contains one more element than the number of rows to publish.

The first element in the array string is a summary that indicates whether errors are detected when parsing the formula, a one-element array of the form is returned; for example:

```
{{"1","Some error message."}}
```

If there are errors during record validation, or if the process is completed successfully, there is one more array element than the number of rows to publish. For example, if there are two rows to publish and both the records have been successfully published, the array string looks like the following example:

```
{{"0",""}, {"0",""}, {"0",""}}
```

If only one record was published successfully, and another failed, then the return array string looks like this:

```
{{"1","An error message"}, {"0",""}, {"1","row level error message"}}
```

Subscription Queries

You can save subscription queries permanently by saving the Excel worksheet containing the formula associated with the query. The next time the Excel Worksheet is opened, the query appears in the Subscription wizard.

Applying a Query

Apply and start a query from the Subscription Wizard pane to populate the Excel worksheet.

Prerequisites

You have defined a query in the Subscription Wizard pane.

Task

1. From the **Subscription Queries** menu, select the desired query.
2. Click **Apply**.
 - The ESP Add-in for Microsoft Excel first verifies that the supplied subscription query name has not already been used then verifies that the provided Start Cell is a valid Excel cell address. If either conditions is false resolve the problem.
 - Next, the ESP Add-in for Microsoft Excel constructs Excel real-time data formulas based on the specified subscription query, and inserts one formula per cell into the active worksheet. Depending on the query, hundreds of formulas may be inserted. The ESP Add-in for Microsoft Excel uses this logic to insert formulas:
 - Formulas are always inserted as a grid, starting at the specified Start Cell location. Each selected column appears in separate but contiguous columns in the Excel worksheet. The value of Max Rows controls the number of rows to which the filter is applied.
 - Soon after the first formula is inserted into the active worksheet, Excel recognizes the real-time data formula and makes a call to the ESP Add-in for Microsoft Excel server that passes the query information for the first filter. The real-time server looks at the information passed, recognizes it as a new query, and spawns a query object. The real-time data server also stores the passed-in information for future use.
 - This process is repeated for every formula of the query, except the real-time server recognizes that the formula is part of the previously seen query. Therefore, it does not create a new query object. Rather, it stores the information so that it can return the data corresponding to the formula.
3. Click **Start**.
 - The ESP Add-in for Microsoft Excel verifies that the connection to Event Stream Processor is active and that the specified query is still valid. If either of these condition is false, then it returns to the formula.

- Next, the ESP Add-in for Microsoft Excel spawns a new read thread to read the transaction log data from Event Stream Processor, and stores it in an internal buffer.
- Every tenth of a second, the ESP Add-in for Microsoft Excel reads the transaction logs from the internal buffer, and decides whether to insert, update, or delete records in a display buffer, based on the user-specified key fields. When there is an insert into the display buffer and the number of records in the buffer is equal to the specified Max Rows, the oldest record in the buffer is deleted, the rest of the records are moved up, and the record is inserted at end. When a record needs to be updated, an in-place update is performed. This insert and update mechanism results in a more stable view of the data in the Excel worksheet, and makes it easier to create charts based on the subscribed data.
- Once the display buffer has been populated, the ESP Add-in for Microsoft Excel notifies Excel that new data is available. When it receives a request for the data, it sends the data in a format that Excel can understand and shows it the appropriate location in the worksheet.

Known Issues and Limitations

The ESP Add-in for Microsoft Excel has some known issues and limitations.

- Performance degrades when Max Rows is set to a large value, for example, several thousand rows or more. The machine becomes very busy as it attempts to process and complete the request.
- When Event Stream Processor stops or the connection is lost due to network failure, the ESP Add-in for Microsoft Excel screen is not automatically refreshed to reflect the current state of the query and connection. Refresh the screen by selecting either the connection or the query to show the current state of the selected object.
- You cannot use more than one worksheet containing ESP Add-in for Microsoft Excel connection or subscription information within the same instance of Excel.

FIX Adapter

Adapter type: fixplugin. The Sybase Event Stream Processor FIX adapter is an implementation of the opensource QuickFIX engine that has been integrated with the Sybase Event Stream Processor API.

The FIX adapter:

- Engages in and manages FIX sessions with well-behaved FIX engines
- Receives and sends FIX messages via connectors and FIX sessions
- Validates inbound FIX messages
- Translates FIX messages into Event Stream Processor records
- Translates Event Stream Processor records into FIX messages

Note: The FIX adapter supports customization of the FIX dictionary.

CHAPTER 2: Adapters Supported by Event Stream Processor

The FIX adapter requires a separately purchased license. This license supports the standard SySAM grace period, meaning it can run unlicensed for 30 days. After this period, the adapter cannot run without a valid license.

If you purchased your product from Sybase or an authorized Sybase reseller, go to the secure Sybase Product Download Center (SPDC) at <https://sybase.subscribenet.com> and log in to generate license keys. The license generation process may vary slightly, depending on whether you ordered directly from Sybase or from a Sybase reseller.

If you ordered your product under an SAP® contract and were directed to download from SAP Service Marketplace (SMP), you can use SMP at <http://service.sap.com/licensekeys> to generate license keys for Sybase products that use SySAM 2-based licenses.

See also

- *File FIX Input Adapter* on page 42
- *File FIX Output Adapter* on page 44
- *Socket FIX Input Adapter* on page 92
- *Socket FIX Output Adapter* on page 94

Supported FIX Versions

FIX protocol versions supported by the FIX adapter.

The FIX adapter supports FIX protocol versions 4.0 through 5.0.

Note: FIXML is not supported.

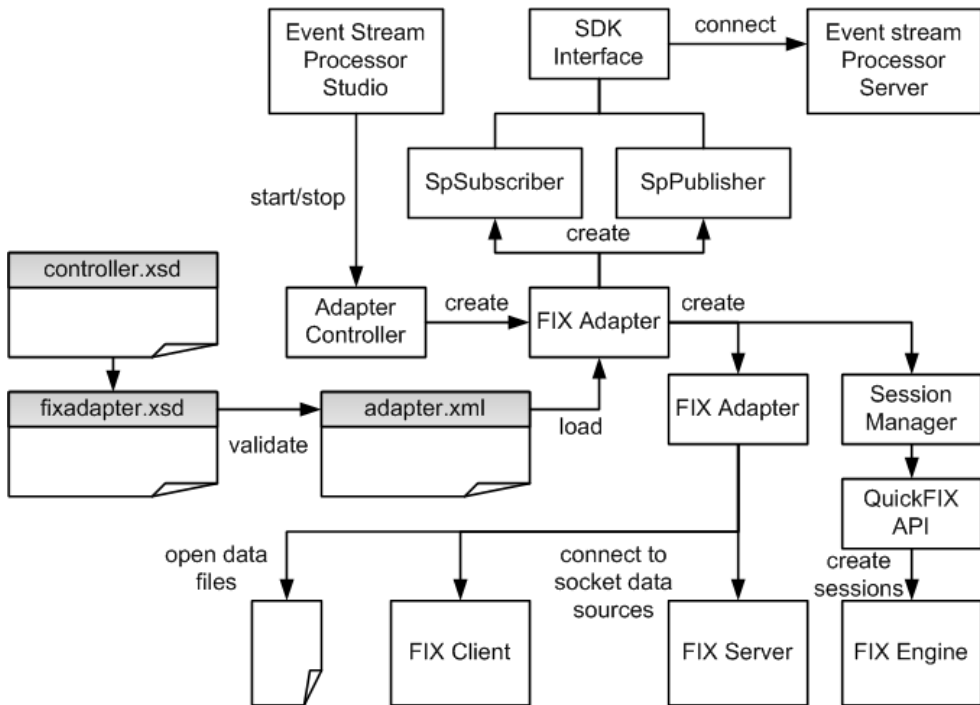
Control Flow

The adapter loads its configuration from a file (for example, `adapter.xml`) and validates it against the adapter schema (`fixadapter.xsd`), which includes the API-wide controller schema (`controller.xsd`).

You cannot edit schemas.

The FIX adapter control flow includes loading different configuration files, and various commands and components.

The Adapter Controller creates an instance of the adapters, and receives and executes user commands. It can execute **start**, **stop**, and **status** commands.

Figure 2: Control Flow

Start Command

The **start** command starts the FIX adapter, configures and starts the command and control interface, loads the FIX dictionary and the SpPublisher and SpSubscriber components, and then connects to the Event Stream Processor via the API interfaces.

The Message Distributor prepares to publish and subscribe to data streams. Data streams are organized into hierarchies named stream clusters. A stream cluster is a set of streams capable of hosting FIX messages of a certain type.

The Connector Manager opens FIX data files and socket connections to client and server sources of FIX data, and the Session Manager uses the QuickFIX API to create and log on to sessions with well-behaved FIX engines. The SpSubscriber and SpPublisher components connect to Event Stream Processor via the API interfaces. SpSubscriber starts listening to output streams, and SpPublisher is ready to publish data to input streams.

The adapter ignores the **start** command if it is executed when there is a running instance of the adapter, and a warning is sent.

See also

- *Data Streams* on page 138
- *Message Flow* on page 142

- *Starting the FIX Adapter* on page 164

Stop Command

The **stop** command causes the datasource Handler to close the session and disconnect from the datasource, the Adapter Controller to stop listening to user commands, and the adapter process to terminate.

The Connector Manager closes any open data files and socket connections to client and server datasources, and the Session Manager logs out of existing sessions.

The adapter ignores the **stop** command if it is executed when there is no running instance of the adapter, and a warning is sent.

See also

- *Stopping the FIX Adapter* on page 166

Status Command

The **status** command reports the FIX adapter status, and the Adapter Controller prints out its status: either running or stopped.

See also

- *Checking the FIX Adapter Status* on page 165

Data Streams

Input FIX messages are stored as stream records organized into stream clusters.

The FIX adapter stores individual messages in multiple records that belong to a stream hierarchy named stream cluster. The top stream in the stream cluster is called the main stream. The main stream stores fields that belong to the FIX message. All the other streams in the stream cluster store fields that belong to nested groups.

Note: Messages of the same type can be stored in more than one stream cluster. These clusters do not have to share a common structure.

Store inbound messages in source streams only and outbound messages in any kind of stream.

The FIX adapter ensures proper indexing of records related to inbound messages. Proper indexing of outbound records is the responsibility of the person creating the model.

The adapter `templates` directory contains generated models for all FIX message types. These automatically generated, exhaustive projects can be used to create stream clusters that serve specific business purposes.

See also

- *Message Flow* on page 142
- *Start Command* on page 137

Example: FIX Input Adapter Data Stream

Sample of a FIX Input adapter data stream.

This is a Quote type FIX message.

```
8=FIX.4.4 | 9=204 | 35=S | 49=COUNTERPARTYA | 55=AASymbol | 117=AAQuoteID |
133=31.1 | 453=2 | 448=AAPartyID1 | 447=B | 452=1 | 802=2 | 523=AAPartySubID11 |
803=1 | 523=AAPartySubID12 | 803=2 | 448=AAPartyID2 | 447=C | 452=2 | 802=1 |
523=AAPartySubID21 | 803=3 | 10=107 |
```

That contains these fields:

- SenderCompID=COUNTERPARTYA (tag 49)
- QuoteID=AAQuoteID (tag 117)
- Symbol=AASymbol (tag 55)
- OfferPx=31.1 (tag 133)
- NoPartyIDs=2 (tag 453)

The message for Event Stream Processor is in this main stream:

```
<SourceStream id="MyQuotes" store="FixStore">
  <Column datatype="string" name="SenderCompID"/>
  <Column datatype="string" name="QuoteID"/>
  <Column datatype="integer" name="NoPartyIDs"/>
  <Column datatype="string" name="Symbol"/>
  <Column datatype="float" name="OfferPx"/>
  <Column datatype="long" name="FixMsgId" key="true"/>
</SourceStream>
```

This is the message for the Server:

```
CREATE MEMORY STORE FixStore PROPERTIES INDEXTYPE='tree',
INDEXSIZEHINT=8;

CREATE INPUT WINDOW MyQuotes
SCHEMA (SenderCompID STRING, QuoteID STRING, NoPartyIDs INTEGER,
Symbol STRING, OfferPx FLOAT, FixMsgId LONG)
PRIMARY KEY (FixMsgId)
STORE FixStore;
```

The message contains two groups of type NoPartyIDs:

Group 1:

- PartyID=AAPartyID1 (tag 448)
- PartyIDSource=B (tag 447)
- PartyRole=1 (Executing Firm, tag 452)
- NoPartySubIDs=2 (tag 802)

Group 2:

CHAPTER 2: Adapters Supported by Event Stream Processor

- PartyID=AAPartyID1 (tag 448)
- PartyIDSource=C (tag 447)
- PartyRole=2 (Broker of Credit, tag 452)
- NoPartySubIDs=1 (tag 802)

Groups 1 and 2 for Event Stream Processor are stored in this stream:

```
<SourceStream id="MyQuotes_NoPartyIDs" store="FixStore">
  <Column datatype="string" name="PartyID"/>
  <Column datatype="string" name="PartyIDSource"/>
  <Column datatype="integer" name="PartyRole"/>
  <Column datatype="integer" name="NoPartySubIDs"/>
  <Column datatype="long" name="FixMsgId" key="true"/>
  <Column datatype="long" name="NoPartyIDs_Num" key="true"/>
</SourceStream>
```

Groups 1 and 2 for the Server are stored in this stream:

```
CREATE INPUT WINDOW MyQuotes_NoPartyIDs
SCHEMA (PartyID STRING, PartyIDSource STRING, PartyRole INTEGER,
NoPartySubIDs INTEGER, FixMsgId LONG, NoPartyIDs_Num LONG)
PRIMARY KEY (FixMsgId, NoPartyIDs_Num)
STORE FixStore;
```

Group 1 and Group 2 contain their own groups of type NoPartySubIDs. Groups 11 and 12 below are part of Group 1:

Group 11:

- PartySubID=AAPartySubID11 (tag 523)
- PartySubIDType=1 (Firm, tag 803)

Group 12:

- PartySubID=AAPartySubID12 (tag 523)
- PartySubIDType=2 (Person, tag 803)

Group 21 is part of Group 2:

- PartySubID=AAPartySubID21 (tag 523)
- PartySubIDType=3 (System, tag 803)

Groups 11, 12, and 21 in Event Stream Processor are stored in this stream:

```
<SourceStream id="MyQuotes_NoPartyIDs_NoPartySubIDs"
store="FixStore">
  <Column datatype="string" name="PartySubID"/>
  <Column datatype="integer" name="PartySubIDType"/>
  <Column datatype="long" name="FixMsgId" key="true"/>
  <Column datatype="long" name="NoPartyIDs_Num" key="true"/>
  <Column datatype="long" name="NoPartySubIDs_Num" key="true"/>
</SourceStream>
```

Groups 11, 12, and 21 for the Server are stored in this stream:

```
CREATE INPUT WINDOW MyQuotes_NoPartyIDs_NoPartySubIDs
SCHEMA (PartySubID STRING, PartySubIDType INTEGER, FixMsgId LONG,
```

```
NoPartyIDs_Num LONG, NoPartySubIDs_Num LONG)
PRIMARY KEY (FixMsgId, NoPartyIDs_Num, NoPartySubIDs_Num)
STORE FixStore;
```

Stream and Column Names

Ensure that the field names of the stream columns correspond to FIX fields. The order of columns does not have to follow the order of fields in the FIX dictionary.

Note: Columns unrelated to hosted FIX messages are not allowed.

The names of main streams can be chosen arbitrarily.

Ensure that descendant streams follow a strict naming convention. Since each descendant stream has a parent stream and corresponds to a repeating group, ensure that its name follows this form:

```
<parent stream name>_<name of the repeating group>
```

Header and Trailer Fields

You can add or update header and trailer fields to create a valid FIX message. Some columns may correspond to header or trailer fields. Output connectors keep all fields in the message body intact, as stored in stream columns.

Record Indexing

FIX messages are stored in a hierarchy of records, and cross-referenced using index columns.

Index columns have a long type, and are located at the end of the stream.

Records in the main stream have only one index. Child records have two indexes, the first must have the same value as the parent record. Descendant records at the next level have three indexes, the first two must have the same value as their parent record, and so on.

Adapters and Sessions

The FIX adapter exchanges FIX messages with datasources such as files and socket connections, as well as other FIX engines.

Files and socket connections are handled by the Connection Manager. Sessions with other FIX engines are handled by the Session Manager.

Note: The adapter can work with any number of different datasources simultaneously.

Adapters are one-directional. Each adapter can be used to receive messages from (or send them to) a single source of FIX data, such as file, client socket connection, or server socket connection. By default, file and socket adapters validate the checksum and body length tags of all input messages. You can turn validation off in the adapter configuration file.

Sessions with other FIX engines are two-directional. Input messages received via sessions are always validated. You cannot turn off validation for sessions. Session management (login, logout, message sequence numbers, resending of messages, and so on), as well as message validation, is performed by the QuickFIX API.

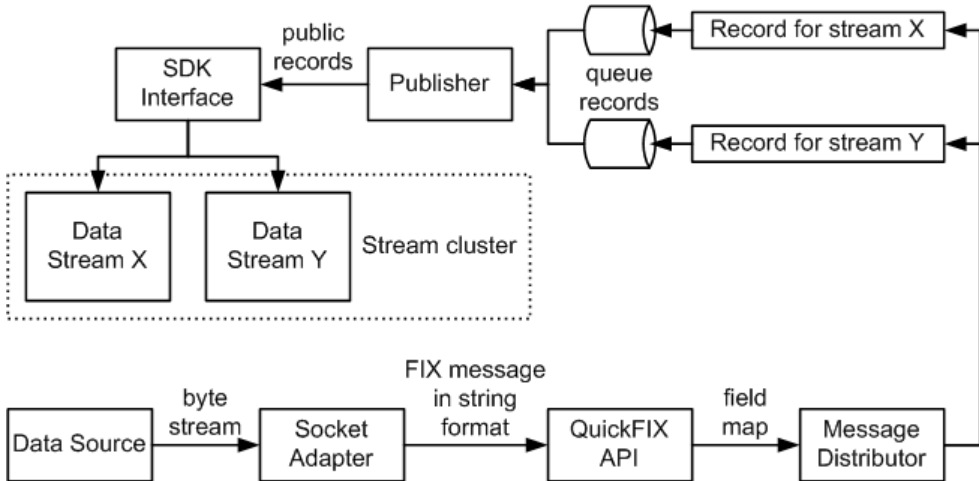
Invalid input messages are discarded and any errors are logged. Otherwise, messages are parsed and published to stream clusters even if their checksum or body length tags are absent or have incorrect values.

Before sending an output message, the adapter recalculates its checksum and body length and updates the appropriate tags.

Message Flow

The message flow through the adapter is initiated by the **start** command.

Figure 3: Inbound Message Flow Through a Socket Connector



The connector receives FIX data as a byte stream. FIX messages are prepared into string objects. The QuickFIX API parses the strings into field maps, and those field maps pass on to the Message Distributor.

The Message Distributor converts each field map into a number of records targeting a stream cluster. The records are now ready to be published to Event Stream Processor. However, they are not published immediately. Records are queued, then picked up by the Publisher object on separate threads, one thread for each record queue. You can configure the queue capacity. A larger queue is less likely to overflow in the event of a message burst. When the queue becomes three-quarters full, a warning is logged. Another warning is logged when the queue returns to three-quarters empty. If the queue is full, the adapter waits until room becomes available before placing the next record.

Records are published asynchronously. The adapter receives no feedback from Event Stream Processor.

If you are using the adapter with Event Stream Processor, in the event of a failover, the SDK interface switches to the spare Event Stream Processor instance without message loss.

See also

- *Data Streams* on page 138
- *Start Command* on page 137

Datatype Mapping for the FIX Adapter

Event Stream Processor datatypes map to FIX datatypes.

Event Stream Processor Datatype	QuickFix Datatype
integer	boolean
string	byte[]
string	char
string	string
date	date
float	float
integer	integer
date or timestamp	UTCDateOnly
date or timestamp	UTCTimeOnly
date or timestamp	UTCTimeStamp

Setting the JAVA_HOME Environment Variable

Set the JAVA_HOME environment variable to point to the Java directory.

Prerequisites

Install Java Runtime Environment version 1.6.0_26 or higher.

Task

Set the JAVA_HOME environment variable to the directory path where Java Runtime Environment 1.6.0_26 or higher is installed.

Next

Verify that the ESP_HOME environment variable is set correctly.

Configuration

Configuration information for the FIX adapter.

FIX Adapter Directory

The adapter directory contains all files, such as configuration files, templates, examples, and JAR files, relating to the adapter.

```

README.txt      Documentation note
ReleaseNotes.txt  Release notes

bin/
  adapter.bat      Standalone adapter startup script
  adapter.sh       Standalone adapter startup script
  adapter-plugin.bat Plug-in connector startup script
  adapter-plugin.sh Plug-in connector startup script

config/
  controller.xsd   Controller schema
  fixadapter.xsd   Adapter schema
  log4j.properties Sample logging configuration
  login.config     Authentication configuration

dictionary/
  FIX40.xml        FIX 4.0 dictionary
  FIX41.xml        FIX 4.1 dictionary
  FIX42.xml        FIX 4.2 dictionary
  FIX43.xml        FIX 4.3 dictionary
  FIX44.xml        FIX 4.4 dictionary

examples/
  AllInOne/
  ClientSocketConnectors/
  FileConnectors/
  ServerSocketConnectors/

lib/
  quickfixj/
    mina-core-1.1.0.jar
    quickfixj-all-1.3.2.jar
    slf4j-api-1.5.6.jar
    slf4j-simple-1.5.6.jar
  sylapij.dll (Windows)
  libsylapij.so (UNIX)
  esp_fix_adapter.jar FIX
templates/
  FlexStream40.xml
  FlexStream41.xml
  FlexStream42.xml
  FlexStream43.xml
  FlexStream44.xml
  SourceStream40.xml
  SourceStream41.xml
  SourceStream42.xml
  SourceStream43.xml
  SourceStream44.xml

```

Schema and Configuration File

The adapter configuration is loaded from a file and validated against the adapter schema.

Ensure that the FIX adapter configuration file is placed into the `$ESP_HOME/adapters/fix/config` before you start the adapter, and that the adapter configuration validates against the schema.

The `$ESP_HOME/adapters/fix/examples` folder contains sample adapter configuration files. You can edit any of these files or write a new one.

Note: The adapter manager looks for either `<sp>` or `<sdk>` node in the configuration file. An `<sp>` node indicates a connection to Event Stream Processor.

Adapter Controller Parameters

The Adapter Controller port listens for commands.

Parameter Name	Type	Description
controllerPort	positive integer	(Required) Specifies the adapter command and control port. User commands are sent to this port on localhost. <hr/> Note: Ensure that each adapter instance has its own dedicated port.

Event Stream Processor Parameters

Event Stream Processor parameters configure communication between Event Stream Processor and the FIX adapter.

These parameters are defined in the `controller.xsd` file in the `config` directory.

Parameter Name	Type	Description
espAuthType	string	(Required) Specifies method used to authenticate to the Event Stream Processor. Valid values are: <ul style="list-style-type: none"> server_rsa – RSA authentication using keystore user_password – Kerberos and LDAP authentication none – No authentication If the adapter is operated as a Studio plug-in, espAuthType is overridden by the Authentication Mode Studio start-up parameter.

Parameter Name	Type	Description
espUser	string	(Required) Specifies user name required to log in to Event Stream Processor. It is required for any authentication scheme other than none (see espAuthType). No default value.
espPassword	string	(Required) Specifies the password required to log in to Event Stream Processor. Required for any authentication scheme other than none (see espAuthType). Includes an "encrypted" attribute indicating whether the espPassword value is encrypted. Default value is false. If set to true, the password value is decrypted using espRSAKeyStore and espRSAKeyStorePassword .
espProjectUri	string	(Required) Specifies the total project Uri to connect to Event Stream Processor cluster. For example, <code>esp://localhost:19011/ws1/p1</code> .
pulseInterval	non-negative integer	(Optional) Specifies time interval, in seconds, during which outbound record changes are coalesced by Event Stream Processor, then received by the adapter as a single event. If not set or set to 0, record changes are received individually as they occur.
espHeartbeatPeriod	positive integer	(Optional) Specifies number, in seconds, that adapter waits before sending the next heartbeat to Event Stream Processor. If Event Stream Processor fails to receive two consecutive heartbeats, all records the adapter publishes are marked stale. Default value is 10.
recordQueueCapacity	positive integer	(Optional) Specifies capacity of the record queues. Default value is 4096.
maxPubPoolSize	positive integer	(Optional) Specifies the maximum size of the record pool. Record pooling allows for faster publication. Default value is 256.

Parameter Name	Type	Description
maxPubPoolTime	positive integer	(Optional) Specifies 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	boolean	(Optional) If set to true, pooled messages are published to Event Stream Processor in transactions. If set to false, they are published in envelopes. Default value is false.
espRSAKeyStore	string	(Dependent required) Specifies the location of the RSA keystore, and is used to decrypt the password value. Required if espAuthType is set to <code>server_rsa</code> , or the encrypted attribute for espPassword is set to true, or both.
espRSAKeyStorePassword	string	(Dependent required) Specifies the keystore password, and is used to decrypt the password value. Required if espAuthType is set to <code>server_rsa</code> , or the encrypted attribute for espPassword is set to true, or both.
espEncryptionAlgorithm	string	(Optional) Used when the encrypted attribute for espPassword is set to true. If left blank, RSA is used as default.

FIX Input Adapter

The FIX Input adapter reads FIX messages from any number of file, socket, and session connectors.

Property Label	Property ID	Type	Description
Adapter Directory Path	baseDir	directory	(Required) Specifies the path to the adapter base directory. No default value. Note: This property is ignored if the Connector Remote Directory Path property is supplied.

Property Label	Property ID	Type	Description
Configuration File Path	configFilePath	configFilename	<p>(Required) Specifies the absolute path to the adapter configuration file. No default value.</p> <hr/> <p>Note: This property is ignored if the Remote Configuration File Path property is supplied.</p>
PropertySet	propertyset	string	<p>(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.</p>
Adapter Remote Directory	remoteBaseDir	string	<p>(Advanced) Specifies the path to the adapter remote base directory (for remote execution only). No default value.</p> <hr/> <p>Note: If this property is supplied, the Connector Directory Path property is ignored.</p>

Property Label	Property ID	Type	Description
Adapter Configuration File Path	remoteConfigFilePath	string	<p>(Advanced) Specifies the absolute path to the adapter configuration file (for remote execution only). No default value.</p> <hr/> <p>Note: If this property is supplied, the configuration file path property is ignored.</p>

Event Stream Processor Server Properties

The Server connection properties are provided as attributes and sub-elements of the <sdk> node in the configuration file.

Parameter Name	Description
server	(Required) Specifies the attribute of the sdk node.
serverHost	(Required) Specifies name of the machine the Server is running on.
serverPort	(Required) Specifies port number the Server is listening on.
serverWorkspace	(Required) Workspace.
serverProject	(Required) Program name.
serverUser	(Optional) User name.
serverPassword	(Optional) Password.

FIX Dictionary

The FIX adapter dictionary contains the definitions of FIX message types, components, and fields.

Parameter Name	Type	Description
fixDictionary	string	(Required) Name of the FIX dictionary file. <hr/> Note: Dictionary files for all supported FIX versions (4.0 through 5.0) are provided in the <code>dictionary</code> folder. <hr/> You can edit the definitions of FIX message types, components, and fields.

Stream Configuration

Use the `streams` section in the configuration file to map FIX message types to stream clusters.

Parameter Name	Type	Description
name	string	(Required) Name of the main stream in a stream cluster. No default value.
messageName	string	(Required) name of a message type hosted on the stream cluster. No default value.

To host FIX messages of type `Quote` in a stream cluster descending from the `MyQuotes` stream, add this fragment to the `<streams>` group:

```
<stream>
  <name>MyQuotes</name>
  <messageName>Quote</messageName>
</stream>
```

Connectors

The `connector` section in the FIX configuration file defines the file and socket connectors.

Parameter Name	Type	Description
streamNames	streamName- Type	(Required) Lists the names of the main streams in stream clusters where messages coming through this connector are hosted.

Inbound and Outbound Connectors

The **inbound** and **outbound** parameter in the FIX configuration file lists inbound and outbound file and socket connectors.

Parameter Name	Type	Description
doValidation	boolean	<p>(Optional) If set to true, inbound messages coming through this connector are validated for correct message length and checksum. If set to false, the message length and checksum fields are ignored. Invalid messages are discarded and the errors are logged. Default value is true.</p> <hr/> <p>Note: Message data is validated against the data dictionary during the parsing of the message.</p> <hr/>

See also

- *Example: Using All In One* on page 173

Sample Configuration File for All In One Connectors

Sample configuration file (adapter.xml) for the all in one connectors in the FIX adapter.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <!-- Sybase ESP FIX adapter configuration file
-->
- <adapter>
- <!-- Adapter Controller
-->
- <controller>
  <controllerPort>13579</controllerPort>
</controller>
- <!-- Event Stream Processor settings
-->
- <esp>
- <espConnection>
  <espProjectUri>esp://localhost:19011/wl/pl</espProjectUri>
</espConnection>
- <espSecurity>
  <espUser>espuser</espUser>
  <espPassword encrypted="false">espuser</espPassword>
  <espAuthType>none</espAuthType>
- <!-- <espRSAKeyStore>/keystore/keystore.jks</
espRSAKeyStore>
  <espRSAKeyStorePassword>Sybase123</espRSAKeyStorePassword>
-->
  <espEncryptionAlgorithm>RSA</espEncryptionAlgorithm>
</espSecurity>
</esp>
```

CHAPTER 2: Adapters Supported by Event Stream Processor

```
- <!-- FIX dictionary
-->
<fixDictionary>FIX44.xml</fixDictionary>
- <!-- Stream cluster to FIX message mapping
-->
- <streams>
- <stream>
  <name>MyQuotes</name>
  <messageName>Quote</messageName>
</stream>
- <stream>
  <name>MyOrders</name>
  <messageName>NewOrderSingle</messageName>
</stream>
</streams>
- <!-- Connectors
-->
- <connectors>
- <outbound>
- <fileConnector>
  <fileName>orders.fix</fileName>
- <streamNames>
  <streamName>MyOrders</streamName>
</streamNames>
</fileConnector>
</outbound>
</connectors>
- <!-- FIX Session Settings
-->
- <sessionSettings>
- <default>
  <ConnectionType>acceptor</ConnectionType>
  <SocketAcceptPort>23456</SocketAcceptPort>
  <FileLogPath>logs</FileLogPath>
  <FileStorePath>store</FileStorePath>
  <DataDictionary>FIX44.xml</DataDictionary>
  <HeartBtInt>600</HeartBtInt>
  <BeginString>FIX.4.4</BeginString>
  <StartTime>00:00:00</StartTime>
  <EndTime>23:59:59</EndTime>
  <SenderCompID>SYBASE</SenderCompID>
</default>
- <sessionSetting>
  <TargetCompID>COUNTERPARTYA</TargetCompID>
</sessionSetting>
- <sessionSetting>
  <TargetCompID>COUNTERPARTYB</TargetCompID>
</sessionSetting>
</sessionSettings>
- <!-- Session logins
-->
- <sessionLogins>
- <senderLogin>
  <username>MyUsername</username>
  <password>MyPassword</password>
  <NextExpectedMsgSeqNum>1</NextExpectedMsgSeqNum>
```

```

</senderLogin>
- <targetLogin>
  <TargetCompID>COUNTERPARTYA</TargetCompID>
  <username>UsernameA</username>
  <password>PasswordA</password>
</targetLogin>
- <targetLogin>
  <TargetCompID>COUNTERPARTYB</TargetCompID>
  <username>UsernameB</username>
  <password>PasswordB</password>
</targetLogin>
</sessionLogins>
- <!-- Sessions
-->
- <sessions>
- <inbound>
- <session>
  <TargetCompID>COUNTERPARTYA</TargetCompID>
- <streamNames>
  <streamName>MyQuotes</streamName>
</streamNames>
</session>
</inbound>
- <outbound>
- <session>
  <TargetCompID>COUNTERPARTYA</TargetCompID>
- <streamNames>
  <streamName>MyOrders</streamName>
</streamNames>
</session>
- <session>
  <TargetCompID>COUNTERPARTYB</TargetCompID>
- <streamNames>
  <streamName>MyQuotes</streamName>
</streamNames>
</session>
</outbound>
</sessions>
</adapter>

```

File Connectors

The **fileConnector** parameter in the FIX configuration file lists property values for file connectors.

Parameter Name	Type	Description
fileName	string	(Required) The relative or absolute path to the file containing FIX data.
streamNames	streamNamesType	(Required) Lists the names of the main streams in stream clusters where messages coming through this connector are hosted.

Parameter Name	Type	Description
doValidation	boolean	<p>(Optional) If set to true, inbound messages coming through this connector are validated for correct message length and checksum. If set to false, the message length and checksum fields are ignored. Invalid messages are discarded and the errors are logged. Default value is true.</p> <hr/> <p>Note: Message data is validated against the data dictionary during the parsing of the message.</p>

See also

- *Example: Using File Connectors* on page 167

Sample Configuration File for File Connectors

Sample configuration file (`adapter.xml`) for the file connectors in the FIX adapter.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <!-- Sybase ESP FIX adapter configuration file
-->
- <adapter>
- <!-- Adapter Controller
-->
- <controller>
  <controllerPort>13579</controllerPort>
</controller>
- <!-- Event Stream Processor settings
-->
- <esp>
  <espConnection>
    <espProjectUri>esp://localhost:19011/w1/p1</espProjectUri>
  </espConnection>
  <espSecurity>
    <espUser>espuser</espUser>
    <espPassword encrypted="false">espuser</espPassword>
    <espAuthType>none</espAuthType>
  <!-- <espRSAKeyStore>/keystore/keystore.jks</
espRSAKeyStore>
    <espRSAKeyStorePassword>Sybase123</espRSAKeyStorePassword>
  -->
  <espEncryptionAlgorithm>RSA</espEncryptionAlgorithm>
</espSecurity>
</esp>
- <!-- FIX dictionary
-->
  <fixDictionary>FIX44.xml</fixDictionary>
- <!-- Stream cluster to FIX message mapping
-->
```

```

- <streams>
- <stream>
  <name>MyQuotes</name>
  <messageName>Quote</messageName>
</stream>
- <stream>
  <name>MyOrders</name>
  <messageName>NewOrderSingle</messageName>
</stream>
</streams>
- <!-- Connectors
-->
- <connectors>
- <inbound>
- <fileConnector>
  <fileName>quotes.fix</fileName>
  <doValidation>>false</doValidation>
- <streamNames>
  <streamName>MyQuotes</streamName>
</streamNames>
</fileConnector>
</inbound>
- <outbound>
- <fileConnector>
  <fileName>orders.fix</fileName>
- <streamNames>
  <streamName>MyOrders</streamName>
</streamNames>
</fileConnector>
</outbound>
</connectors>
- <!-- Sessions
-->
  <sessions />
</adapter>

```

Client Socket Connectors

The **clientSocketConnector** parameters in the FIX configuration file define the name, IP address, validation scheme, stream names, and port of the Server to send FIX messages to.

Parameter Name	Type	Description
dataHost	string	(Required) Specifies the name or IP address of the server to send FIX messages to.
dataPort	nonNegativeInteger	(Required) Specifies the port to connect to.
streamNames	streamName- Type	(Required) Lists the names of the main streams in stream clusters where messages coming through this connector are hosted.

Parameter Name	Type	Description
doValidation	boolean	<p>(Optional) If set to true, inbound messages coming through this connector are validated for correct message length and checksum. If set to false, the message length and checksum fields are ignored. Invalid messages are discarded and the errors are logged. Default value is true.</p> <hr/> <p>Note: Message data is validated against the data dictionary during the parsing of the message.</p>

See also

- *Example: Using Client Socket Connectors* on page 168

Sample Configuration File for Client Socket Connectors

Sample configuration file (adapter.xml) for the client socket connectors in the FIX adapter.

```

<?xml version="1.0" encoding="UTF-8" ?>
- <!-- Sybase ESP FIX adapter configuration file
-->
- <adapter>
- <!-- Adapter Controller
-->
- <controller>
  <controllerPort>13579</controllerPort>
</controller>
- <!-- Event Stream Processor settings
-->
- <esp>
- <espConnection>
  <espProjectUri>esp://localhost:19011/wl/pl</espProjectUri>
</espConnection>
- <espSecurity>
  <espUser>espuser</espUser>
  <espPassword encrypted="false">espuser</espPassword>
  <espAuthType>none</espAuthType>
- <!-- <espRSAKeyStore>/keystore/keystore.jks</
espRSAKeyStore>
  <espRSAKeyStorePassword>Sybase123</espRSAKeyStorePassword>
-->
  <espEncryptionAlgorithm>RSA</espEncryptionAlgorithm>
</espSecurity>
</esp>
- <!-- FIX dictionary
-->
  <fixDictionary>FIX44.xml</fixDictionary>
- <!-- Stream cluster to FIX message mapping

```

```

-->
- <streams>
- <stream>
  <name>MyQuotes</name>
  <messageName>Quote</messageName>
</stream>
- <stream>
  <name>MyOrders</name>
  <messageName>NewOrderSingle</messageName>
</stream>
</streams>
- <!-- Connectors
-->
- <connectors>
- <inbound>
- <clientSocketConnector>
  <dataHost>localhost</dataHost>
  <dataPort>43210</dataPort>
  <doValidation>true</doValidation>
- <streamNames>
  <streamName>MyQuotes</streamName>
</streamNames>
</clientSocketConnector>
</inbound>
- <outbound>
- <clientSocketConnector>
  <dataHost>localhost</dataHost>
  <dataPort>32109</dataPort>
- <streamNames>
  <streamName>MyOrders</streamName>
</streamNames>
</clientSocketConnector>
</outbound>
</connectors>
- <!-- Sessions
-->
  <sessions />
</adapter>

```

Server Socket Connectors

The **serverSocketConnector** parameter in the FIX configuration file defines the port on which the Server is listening for client connections.

Parameter Name	Type	Description
dataPort	nonNegativeInteger	(Required) Specifies the port the server is listening for client connections.
streamNames	streamName- Type	(Required) Lists the names of the main streams in stream clusters where messages coming through this connector are hosted.

Parameter Name	Type	Description
doValidation	boolean	<p>(Optional) If set to true, inbound messages coming through this connector are validated for correct message length and checksum. If set to false, the message length and checksum fields are ignored. Invalid messages are discarded and the errors are logged. Default value is true.</p> <hr/> <p>Note: Message data is validated against the data dictionary during the parsing of the message.</p>

See also

- *Example: Using Server Socket Connectors* on page 170

Sample Configuration File for Server Socket Connectors

Sample configuration file (`adapter.xml`) for the server socket connectors in the FIX adapter.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <!-- Sybase ESP FIX adapter configuration file
-->
- <adapter>
- <!-- Adapter Controller
-->
- <controller>
  <controllerPort>13579</controllerPort>
</controller>
- <!-- Event Stream Processor settings
-->
- <esp>
- <espConnection>
  <espProjectUri>esp://localhost:19011/w1/p1</espProjectUri>
</espConnection>
- <espSecurity>
  <espUser>espuser</espUser>
  <espPassword encrypted="false">espuser</espPassword>
  <espAuthType>none</espAuthType>
- <!-- <espRSAKeyStore>/keystore/keystore.jks</
espRSAKeyStore>
  <espRSAKeyStorePassword>Sybase123</espRSAKeyStorePassword>
-->
  <espEncryptionAlgorithm>RSA</espEncryptionAlgorithm>
</espSecurity>
</esp>
- <!-- FIX dictionary
-->
  <fixDictionary>FIX44.xml</fixDictionary>
```



```

- <!-- Stream cluster to FIX message mapping
-->
- <streams>
- <stream>
  <name>MyQuotes</name>
  <messageName>Quote</messageName>
</stream>
- <stream>
  <name>MyOrders</name>
  <messageName>NewOrderSingle</messageName>
</stream>
</streams>
- <!-- Connectors
-->
- <connectors>
- <inbound>
- <serverSocketConnector>
  <dataPort>54321</dataPort>
  <doValidation>true</doValidation>
- <streamNames>
  <streamName>MyQuotes</streamName>
</streamNames>
</serverSocketConnector>
</inbound>
- <outbound>
- <serverSocketConnector>
  <dataPort>43210</dataPort>
- <streamNames>
  <streamName>MyOrders</streamName>
</streamNames>
</serverSocketConnector>
</outbound>
</connectors>
- <!-- Sessions
-->
  <sessions />
</adapter>

```

Session Settings

Properties in the `sessionSettings` section configure default and specific settings for session connections with the FIX engine.

Default Settings

Properties used to configure default settings for all session connections.

Property Name	Type	Description
ConnectionType	string	<p>(Required) Specifies whether the adapter acts as a server or a client. Valid values are:</p> <ul style="list-style-type: none"> • Acceptor – The adapter acts as a server accepting connection requests from FIX session initiators. • Initiator – The adapter acts as a client connecting to FIX session acceptors. <p>No default value.</p> <hr/> <p>Note: Each adapter instance operates in either acceptor mode or initiator mode, but cannot operate in both modes simultaneously.</p>
SocketAcceptPort	nonNegativeInteger	<p>(Required) Specifies the port on which the adapter listens for connections from FIX session initiators. No default value.</p> <p>Operates only in initiator mode.</p>
FileLogPath	string	<p>(Required) Specifies the directory path for message logs. Both absolute and relative paths are accepted. No default value.</p>
FileStorePath	string	<p>(Required) Specifies the directory path for message stores. Both absolute and relative paths are accepted. No default value.</p>
StartTime	string	<p>(Required) Specifies the time of the day when the FIX session is activated. No default value.</p>
EndTime	string	<p>(Required) Specifies the time of the day when the FIX session is deactivated. No default value.</p>
DataDictionary	string	<p>(Required) Specifies the absolute or relative paths to the FIX dictionary file path. No default value.</p>
BeginString	string	<p>(Required) Specifies the value of the BeginString field (tag 8) in outbound FIX messages. No default value.</p>

Property Name	Type	Description
SenderCompID	string	(Required) Specifies the value of the SenderCompID field (tag 49) in outbound FIX messages. An adapter instance uses the same SenderCompID value for all session connections.
HeartBtInt	nonNegativeInteger	(Optional) Specifies the heartbeat interval, in seconds. Default value is 10. Operates only in initiator mode.
ReconnectInterval	positiveInteger	(Optional) Specifies the interval between reconnection attempts, in seconds. The adapter keeps trying to reconnect if it fails to connect to the acceptor engine at startup or if the connection is lost afterward. Default value is 30. Operates only in initiator mode.
LogonTimeout	nonNegativeInteger	(Optional) Specifies the number of seconds to wait for a logon response before disconnecting from the acceptor engine. Default value is 10. Operates only in initiator mode.
LogoutTimeout	nonNegativeInteger	(Optional) Specifies the number of seconds to wait for a logout response before disconnecting from the acceptor engine. Default value is 2. Operates only in initiator mode.

Specific Settings

Properties used to configure specific settings for specific session connections.

Property Name	Type	Description
TargetCompID	string	(Required) Specifies the value of the TargetCompID field (tag 56) in outbound FIX messages.

Property Name	Type	Description
SocketConnectHost	string	(Required) Specifies the acceptor engine's host name or IP address. Note: Operates only in initiator mode.
SocketConnectPort	non-negative integer	(Required) Specifies the port on which the acceptor engine is listening for connections. Note: Operates only in initiator mode.

Session Logins

Properties in the `sessionLogins` section configure sender and target login properties for session connections with the FIX engine.

Sender Login Properties

If specified, the adapter attaches **senderLogin** properties to outbound login messages at the beginning of FIX sessions.

Parameter Name	Type	Description
username	string	(Required) Specifies the sender's user name.
password	string	(Required) Specifies the sender's password.
NextExpectedMsgSeqNum	integer	(Required) Specifies the value of the NextExpectedMsgSeqNum field (tag 789) in outbound login messages.

Target Login Properties

For each inbound login message, the FIX adapter tries to match the values of the **username** and **password** fields with the ones specified for the corresponding **TargetCompID** field.

Note: An error is logged if the user names or the passwords do not match.

Parameter Name	Type	Description
username	string	(Required) Specifies the target's user name.
password	string	(Required) Specifies the target's password.
TargetCompID	string	(Required) Specifies the value of the TargetCompID field (tag 56) in inbound login messages.

Session Properties

Properties in **sessionProperties** sections identify inbound and outbound FIX sessions and indicate the main streams of the stream clusters that host data exchanged during FIX sessions.

Property Name	Type	Description
TargetCompID	string	<p>(Required) The session identifier. No default value.</p> <hr/> <p>Note: An inbound and an outbound session with the same identifier are implemented as a single two-way session.</p>
streamNames	streamNamesType	<p>(Required) Specifies the lists the names of the main streams in stream clusters where messages exchanged over this FIX session are hosted.</p> <ul style="list-style-type: none"> For inbound sessions, unmapped messages are ignored; mapped messages are written to all stream clusters as they are mapped to. For outbound sessions, the header and trailer fields are added or updated, if necessary, to ensure the validity of the outgoing FIX messages. <hr/> <p>Note: Two stream clusters hosting messages of the same type are not required to share the same structure.</p>

Example: Receiving and Hosting Inbound Messages

Receive inbound messages from a specified target, and host it in a specified stream cluster with a specified main stream.

Inbound messages received over a FIX session from a target identified as COUNTERPARTYA are hosted in a stream cluster for which MyQuotes is the main stream. Messages of types other than Quote are ignored.

```
<inbound>
  <session>
    <TargetCompID>COUNTERPARTYA</TargetCompID>
    <streamNames>
      <streamName>MyQuotes</streamName>
    </streamNames>
  </session>
</inbound>
```

Logging

The FIX adapter uses Apache log4j API to log errors, warnings, and information and debugging messages.

A sample log4j.properties file containing the logging configuration is part of the FIX adapter distribution. This file is located in the \$ESP_HOME/adapters/fix/examples/<example name>/log4j.properties folder.

Note: Setting the logging value to DEBUG may result in large log files. The default value is INFO.

Refer to <http://logging.apache.org/log4j> for details on logging configuration.

Operation

Operate the FIX adapter from the command line.

To allow the adapter.xml configuration to be placed in any desired location, ensure that the full file path appears along with the **start**, **stop**, and **status** commands.

Note: You can define long file path names as environment variables.

Starting the FIX Adapter

To start the FIX adapter from the command line, start Event Stream Processor, verify parameters, and execute the **start** command.

1. Start Event Stream Processor.

Windows:

1. Start the example cluster.

```
cd %ESP_HOME%\cluster\nodes\node1
%ESP_HOME%\bin\esp_server --cluster-node node1.xml
```

2. Compile CCL to create CCX.

```
%ESP_HOME%\bin\esp_compiler -i model.ccl -o model.ccx
```

3. Deploy the project on the cluster.

```
%ESP_HOME%\bin\esp_cluster_admin" --uri=esp://localhost:19011
--username=sybase --password=sybase --add_project --workspace-
name=w1 --project-name=p1 --ccx=model.ccx
```

4. Start the deployed project on the cluster.

```
%ESP_HOME%\bin\esp_cluster_admin" --uri=esp://localhost:19011
--username=sybase --password=sybase --start_project --
workspace-name=w1 --project-name=p1
```

UNIX:

1. Start the example cluster.

```
cd $ESP_HOME/cluster/nodes/node1
$ESP_HOME/bin/esp_server --cluster-node node1.xml
```

2. Compile CCL to create CCX.

```
$ESP_HOME/bin/esp_compiler -i model.ccl -o model.ccx
```

3. Deploy the project on the cluster.

```
$ESP_HOME/bin/esp_cluster_admin" --uri=esp://localhost:19011
--username=sybase --password=sybase --add_project --workspace-
name=w1 --project-name=p1 --ccx=model.ccx
```

4. Start the deployed project on the cluster.

```
$ESP_HOME/bin/esp_cluster_admin" --uri=esp://localhost:19011
--username=sybase --password=sybase --start_project --
workspace-name=w1 --project-name=p1
```

2. Start the adapter.

Operating System	Step
UNIX	Open a terminal window and enter: <pre>cd \$ESP_HOME/adapters/fix/bin ./adapter.sh <configuration file path> start</pre>
Windows	Open a command window and enter: <pre>cd %ESP_HOME%/adapters/fix/bin adapter.bat <configuration file path> start</pre>

You can use the **esp_subscribe** utility to ensure that FIX messages are successfully published to Event Stream Processor.

See also

- *Start Command* on page 137

Checking the FIX Adapter Status

Run the **status** command in a terminal or command window to check adapter status.

Check the FIX adapter status:

Operating System	Step
UNIX	Open a terminal window and enter: <pre>cd \$ESP_HOME/adapters/fix/bin ./adapter.sh <configuration file path> status</pre>

Operating System	Step
Windows	Open a command window and enter: <pre>cd %ESP_HOME%/adapters/fix/bin adapter.bat <configuration file path> status</pre>

You see the adapter status, which is either running or stopped.

See also

- *Status Command* on page 138

Stopping the FIX Adapter

Run the **stop** command in a terminal or command window to stop the adapter.

Operating System	Step
UNIX	Open a terminal window and enter: <pre>cd \$ESP_HOME/adapters/fix/bin ./adapter.sh \$ADAPTER stop &</pre>
Windows	Open a command window and enter: <pre>cd %ESP_HOME%/adapters/fix/bin adapter.bat %ADAPTER% stop</pre>

See also

- *Stop Command* on page 138

Examples

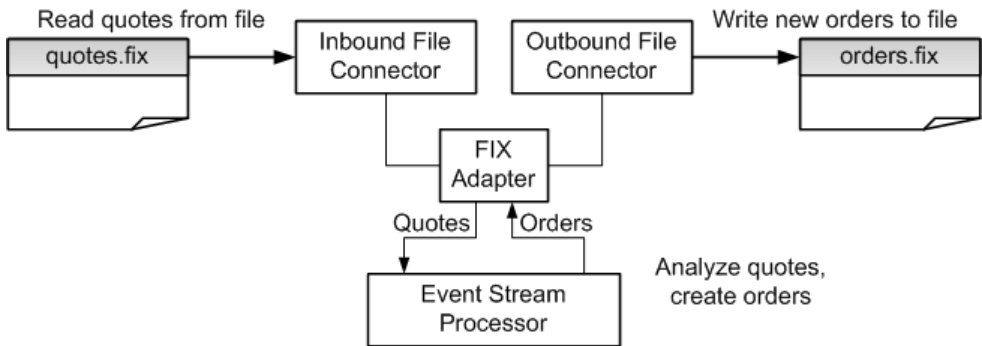
Use the working examples provided with the adapter to learn how to subscribe to real-time market data and publish data to the Event Stream Processor.

The scripts provided with the examples do not require a network connection.

Example: Using File Connectors

Use file connectors to read Quote messages from a file, and publish them to the Event Stream Processor. If the value of the OfferPx field is less than 30.0, write a NewOrderSingle message to another file.

Figure 4: File Connectors



1. Start Event Stream Processor.

Operating System	Step
UNIX	Open a terminal window: 1. Start the example cluster: <code>start_server_cluster.sh</code> 2. Start the project on the cluster: <code>start_project.sh</code>
Windows	Open a command window: 1. Start the example cluster: <code>start_server_cluster.bat</code> 2. Add project to the cluster, and start it on the cluster: <code>start_project.bat</code>

2. Start the respective subscriber utility for Event Stream Processor.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./esp-subscribe.sh</code>
Windows	Open a command window and enter: <code>esp-subscribe.bat</code>

3. Start the adapter.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./adapter.sh</code>
Windows	Open a command window and enter: <code>adapter.bat</code>

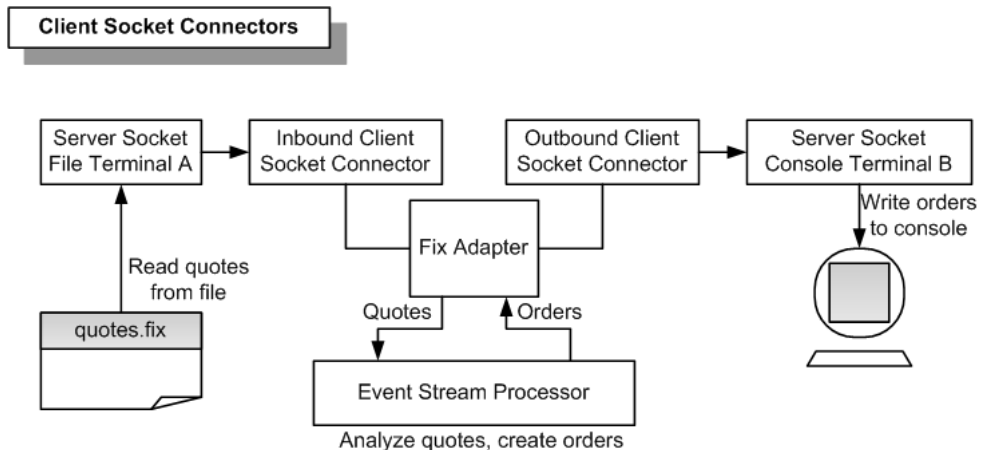
- Wait five to ten seconds for the adapter to initialize.
 - The MyQuotes stream now contains two records. The MyOrders stream contains one record. Use the Event Stream Processor subscriber utility to view the content of the streams.
 - The `orders.fix` file now contains a `NewOrderSingle` message.

See also

- File Connectors* on page 153

Example: Using Client Socket Connectors

Use client socket connectors to read Quote messages from a server socket, and publish them to the Event Stream Processor. If the value of the OfferPx field is less than 30.0, the adapter writes a `NewOrderSingle` message to another server socket.



- Start Event Stream Processor.

Operating System	Step
UNIX	Open a terminal window: <ol style="list-style-type: none"> 1. Start the example cluster: <code>start_server_cluster.sh</code> 2. Start the project on the cluster: <code>start_project.sh</code>
Windows	Open a command window: <ol style="list-style-type: none"> 1. Start the example cluster: <code>start_server_cluster.bat</code> 2. Add project to the cluster, and start it on the cluster: <code>start_project.bat</code>

2. Start the respective subscriber utility for Event Stream Processor.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./esp-subscribe.sh</code>
Windows	Open a command window and enter: <code>esp-subscribe.bat</code>

3. Start the adapter.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./adapter.sh</code>
Windows	Open a command window and enter: <code>adapter.bat</code>

4. Wait five to ten seconds for the adapter to initialize.
5. Start server socket terminal B.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./terminalB.sh</code>
Windows	Open a command window and enter: <code>terminalB.bat</code>

6. Start server socket terminal A.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./terminalA.sh</code>
Windows	Open a command window and enter: <code>terminalA.bat</code>

- The MyQuotes stream now contains two records, and the MyOrders stream contains one record. Use the Event Stream Processor subscriber utility to view the content of the streams.
- The terminal B console window now contains a NewOrderSingle message.

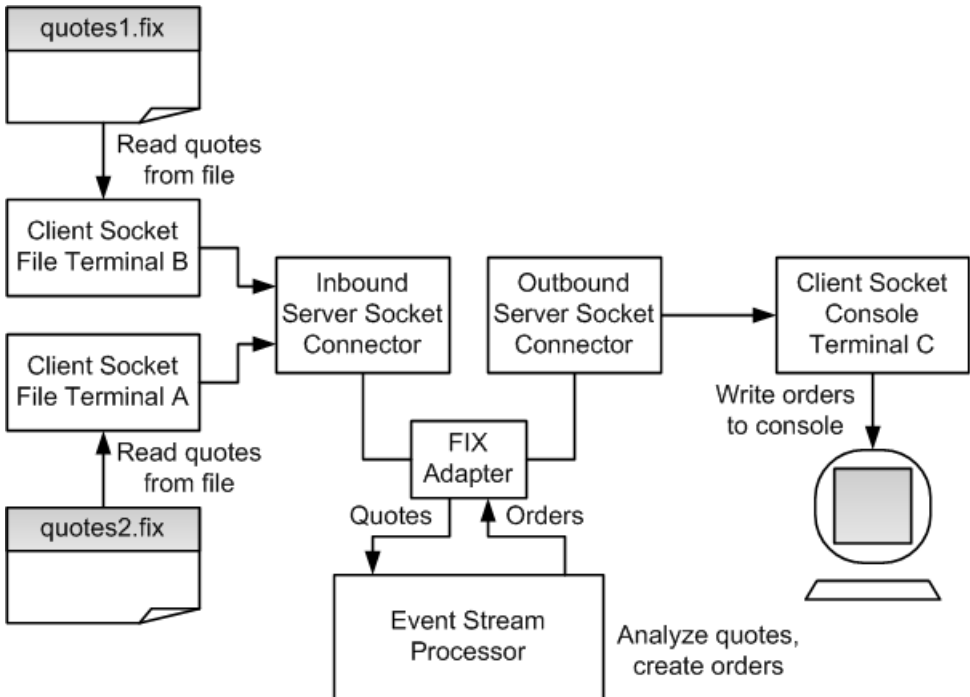
See also

- *Client Socket Connectors* on page 155

Example: Using Server Socket Connectors

Use server socket connectors to read Quote messages from two client sockets, and publish them to the Event Stream Processor. If the value of the OfferPx field is less than 30.0, the FIX adapter writes a NewOrderSingle message to the a third client socket.

Figure 5: Server Socket Connectors



1. Start Event Stream Processor.

Operating System	Step
UNIX	Open a terminal window: 1. Start the example cluster: <code>start_server_cluster.sh</code> 2. Start the project on the cluster: <code>start_project.sh</code>
Windows	Open a command window: 1. Start the example cluster: <code>start_server_cluster.bat</code> 2. Add project to the cluster, and start it on the cluster: <code>start_project.bat</code>

2. Start the respective subscriber utility for Event Stream Processor.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./esp-subscribe.sh</code>
Windows	Open a command window and enter: <code>esp-subscribe.bat</code>

3. Start the adapter.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./adapter.sh</code>
Windows	Open a command window and enter: <code>adapter.bat</code>

4. Wait five to ten seconds for the adapter to initialize.

5. Start output terminal C.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./terminalC.sh</code>
Windows	Open a command window and enter: <code>terminalC.bat</code>

6. Start output terminal B.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./terminalB.sh</code>
Windows	Open a command window and enter: <code>terminalB.bat</code>

7. Start output terminal A.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./terminalA.sh</code>
Windows	Open a command window and enter: <code>terminalA.bat</code>

- The MyQuotes stream now contains three records. The MyOrders stream contains one record. Use the Event Stream Processor subscriber utility to view the content of the streams.
- The terminal C console window now contains a NewOrderSingle message.

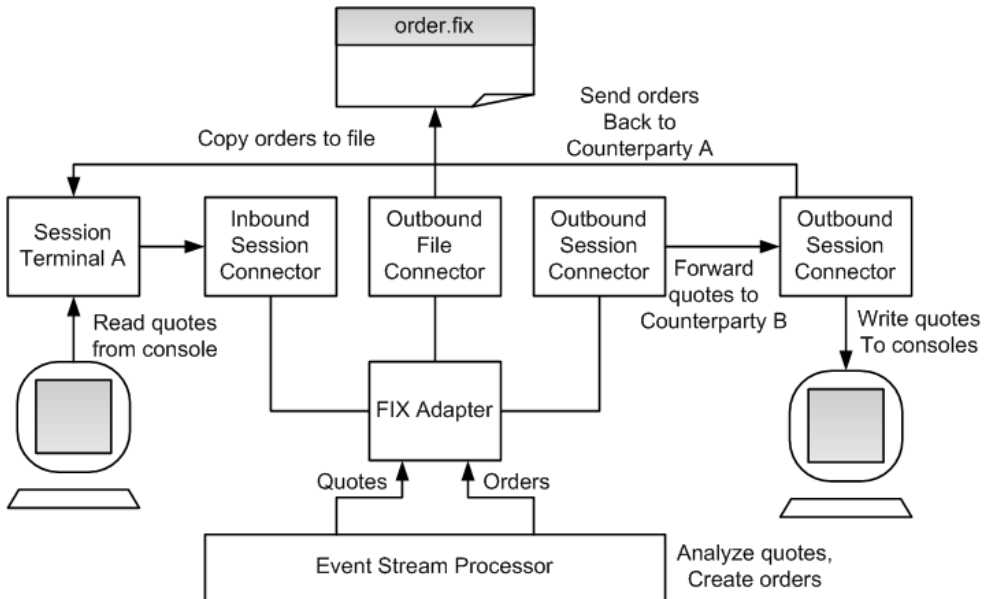
See also

- *Server Socket Connectors* on page 157

Example: Using All In One

Engage in a two-way FIX session with counterparty A, and in a one-way FIX session with counterparty B. If the value of the OfferPx field is less than 30.0, the FIX adapter sends a NewOrderSingle message back to counterparty A and copies it to a file.

Figure 6: All In One



1. Start Event Stream Processor.

Operating System	Step
UNIX	Open a terminal window: 1. Start the example cluster: <code>start_server_cluster.sh</code> 2. Start the project on the cluster: <code>start_project.sh</code>
Windows	Open a command window: 1. Start the example cluster: <code>start_server_cluster.bat</code> 2. Add project to the cluster, and start it on the cluster: <code>start_project.bat</code>

2. Start the respective subscriber utility for Event Stream Processor.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./esp-subscribe.sh</code>
Windows	Open a command window and enter: <code>esp-subscribe.bat</code>

3. Start the adapter.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./adapter.sh</code>
Windows	Open a command window and enter: <code>adapter.bat</code>

4. Wait five to ten seconds for the adapter to initialize.
5. Start output terminal B.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./terminalB.sh</code>
Windows	Open a command window and enter: <code>terminalB.bat</code>

6. Wait five to ten seconds for the adapter to initialize.
7. Start output terminal A.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./terminalA.sh</code>
Windows	Open a command window and enter: <code>terminalA.bat</code>

8. Wait five to ten seconds for the adapter to initialize.
9. Copy the contents of the `quotes.fix` file and paste it into the terminal A console window.

- The MyQuotes stream now contains two records. The MyOrders stream contains 1 record. Use the Event Stream Processor subscriber utility to view the content of the streams.
- The terminal B console window now contains a NewOrderSingle message.
- The `orders.fix` file now contains a copy of the NewOrderSingle message.

See also

- *Inbound and Outbound Connectors* on page 151

Flex Adapter

The Sybase Event Stream Processor Flex Output adapter is a software interface that allows you to obtain data from streams in an Event Stream Processor project and provide it to a full range of Adobe Flex applications.

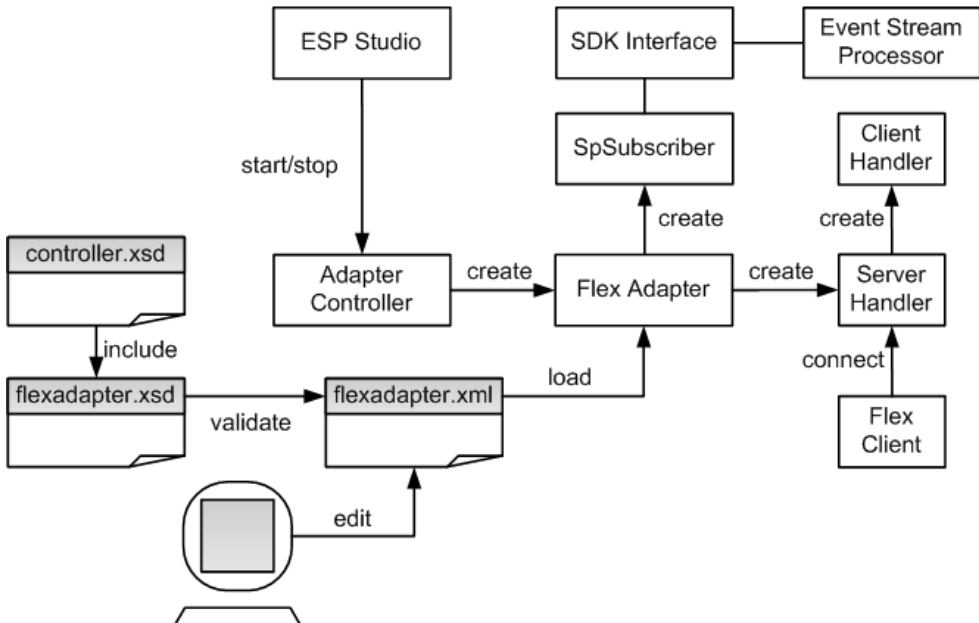
The Flex adapter :

- Runs an internal Flex server, listens and accepts client connections
- Parses client requests and subscribes to streams
- Filters stream records, converts them into XML, and sends them to clients, one record at a time

Control Flow

The Flex adapter loads its configuration from a file (for example, `adapter-pubsub.xml`), and validates it against the adapter schema (`flexadapter.xsd`), which includes the API-wide controller schema (`controller.xsd`).

Figure 7: Flex Adapter Control Flow



CHAPTER 2: Adapters Supported by Event Stream Processor

The Adapter Controller creates an instance of the adapter, and receives and executes **start**, **stop**, and **status** commands.

Start Command

The **start** command configures and starts the control interface, gets the Server Handler to start listening for client connections, creates a separate Client Handler to serve each client connection, and connects the SpSubscriber component to Event Stream Processor via the SDK interface.

The adapter ignores the **start** command if it is executed when there is a running instance of the adapter, and a warning is sent.

See also

- *Starting the Flex Adapter* on page 184

Stop Command

The **stop** command disconnects the SpSubscriber from Event Stream Processor, stops the Server Handler from accepting new client connections, causes the Client Handlers to finalize the responses to existing clients and disconnects them, and terminates the adapter process.

If the **stop** command is executed when there is no instance of a running adapter, the command is ignored and a warning is sent.

See also

- *Stopping the Flex Adapter* on page 186

Status Command

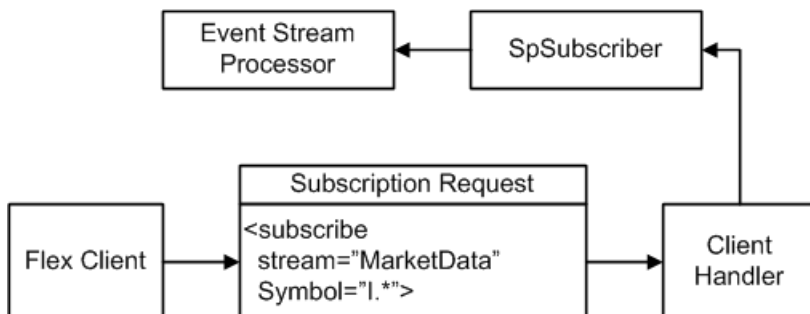
The **status** command reports the adapter status, and the Adapter Controller prints out its status: either running or stopped.

See also

- *Checking the Flex Adapter Status* on page 185

Message Flow

The message flow between the adapter and any Flex client is initiated when the client sends a subscribe request indicating the stream name and a column filter (optional).



Ensure the request has the format:

```
<subscribe stream="MyStream" myFilterColumn1="MyRegex1" .../>
```

Ensure the filtered columns are string type. Regular expressions are accepted as column values. For example, as a result of this request:

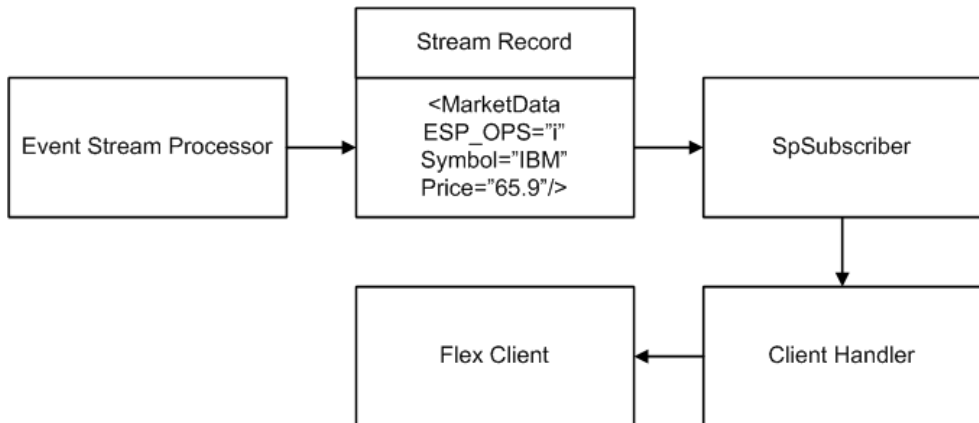
```
<subscribe stream="MarketData" Symbol="I.*"/>
```

the client receives all records on the MarketData stream in which the Symbol begins with the capital letter "I".

The records sent back to the clients have the format:

```
<MyStream ESP_OPS="i" myColumn1="MyValue1" myColumn2="MyValue2" .../
>
```

where ESP_OPS is the record opcode. The valid opcode values are "i" (INSERT), "u" (UPDATE), "d" (DELETE), and "p" (UPSERT). All columns with non-null values are included, regardless of the opcode. Null column values are ignored.



To unsubscribe from a previously subscribed stream, the client sends a request with the format:

```
<unsubscribe stream="MyStream"/>
```

The client can subscribe concurrently to any number of different streams. To change the column filter values, unsubscribe from the stream first, then subscribe with the new filter.

In the event of an Event Stream Processor failover, the SDK API switches to the spare Event Stream Processor instance without message loss.

Stream Handler

Use a Stream Handler for client-server communication.

Although Flex clients can use raw XML to subscribe to and receive stream data from Event Stream Processor, Sybase recommends you delegate the client-server communication to a

CHAPTER 2: Adapters Supported by Event Stream Processor

Stream Handler. To use the Stream Handler functionality, include the `streamhandler.swc` library, located in the `lib` directory, in your Flex client build.

Here is an example of using the Stream Handler in the ActionScript code:

```
import com.sybase.esp.adapter.flex.StreamHandler;
private var streamHandler:StreamHandler = new StreamHandler(
    "localhost", 23456, onConnect, onDisconnect, onRecord);
private function onConnect(event:Event):void {
    // Invoked after the client has successfully connected to
    // the adapter
}
private function onDisconnect(event:Event):void {
    // Invoked after the client has disconnected from
    // the adapter
}
private function onRecord(streamName:String, opcode:String,
    record:Object):void {
    // Invoked when the client receives a record from
    // the adapter
}
```

To subscribe to a stream, invoke the `subscribe()` method of the Stream Handler with the stream name and filter parameters. The filter is an ActionScript object with several properties, one for each filtered column. You can use regular expressions as property values. For example:

```
var filter:Object = new Object();
filter.Symbol = "I.*";
streamHandler.subscribe("Stream1", filter);
```

Ensure that filtered columns are string type. To receive all stream records, without filtering, do not add any properties to the filter object. To unsubscribe from a stream, invoke the `unsubscribe()` method of the Stream Handler, passing in the stream name as a parameter.

For example:

```
streamHandler.unsubscribe("Stream1");
```

Implement the `onRecord()` callback method to process the records coming on a subscription. The callback has three parameters:

- **streamName**
- **Opcode**
- An object which contains all non-null column values as properties

For example:

```
private function onRecord(streamName:String, opcode:String,
    record:Object):void {
    trace("Record received on stream " + streamName);
    trace("Opcode=" + opcode);
    trace("Column values:");
    for (var column:String in record)
        trace(column + "=" + record[column]);
}
```

Setting the JAVA_HOME Environment Variable

Set the JAVA_HOME environment variable to point to the Java directory.

Prerequisites

- Install Java Runtime Environment version 1.6.0_26 or higher. To see if you have a suitable version of Java installed, go to [http://www.java.com/en/download/installed.jsp?detect=jre"&"try=1](http://www.java.com/en/download/installed.jsp?detect=jre).
- To download and install Java, go to [http://jdk.sun.com/webapps/getjava/BrowserRedirect?locale=en"&"host=www.java.com:80](http://jdk.sun.com/webapps/getjava/BrowserRedirect?locale=en).

Task

Set the JAVA_HOME environment variable to the directory path where Java Runtime Environment 1.6.0_26 or higher is installed.

Next

Verify that the ESP_HOME environment variable is set correctly.

Configuration

Configuration information for the Flex adapter.

Flex Adapter Directory

The adapter directory contains all files, such as configuration files, templates, examples, and JAR files, relating to the adapter.

```

README.txt   Quick Guide
ReleaseNotes.txt  Release Notes

bin/
  adapter.bat   Standalone adapter startup script
  adapter.sh    Standalone adapter startup script
  adapter-plugin.bat  Plug-in connector startup script
  adapter-plugin.sh  Plug-in connector startup script

config/
  controller.xsd   Controller schema
  log4j.properties  Sample logging configuration
  flexadapter.xsd  Flex Adapter schema
  login.config     Authentication configuration

example/          Working example

lib/
  esp_flex_adapter.jar  Flex adapter library

javadoc/
  adapterapi/      Adapter API Javadoc
  flexadapter/     Flex Adapter Javadoc

```

Common jars are located here:

```
$ESP_HOME/adapters/jar
```

The directory structure is:

```
activation.jar           Java mail library
adapterapi.jar          Adapter API code
axis.jar                Webservices jar
commons-codec-1.3.jar   Required by SDK API
commons-discovery-0.2.jar
commons-logging-1.1.jar Logging library
esp_java_sdk-0.2.jar    ESP SDK library
jaxrpc-api-1.1.jar     Required by ESP SDK
log4j-1.2.14.jar       Logging library
mail.jar                Java mail library
saaj-api-1.3.jar        Webservices jar
ws-commons-util-1.0.2.jar Required by ESP SDK
wsdl4j-1.5.1.jar
xercesImpl.jar          XML parser library
xmlrpc-client-3.1.3.jar Required by ESP SDK
xmlrpc-common-3.1.3.jar Required by ESP SDK
xmlrpc-server-3.1.3.jar Required by ESP SDK
```

Schema and Configuration File

The adapter configuration loads from a file and validates against the adapter schema.

The example folder contains the `adapter.xml` sample adapter configuration file.

You must provide a valid configuration file and ensure the adapter configuration validates against the adapter schema.

Adapter Controller Parameter

The Adapter Controller parameter specifies the adapter command and control port.

This parameter is defined in the `controller.xsd` file in the `config` directory.

Parameter Name	Type	Description
<code>controllerPort</code>	positive integer	(Required) Specifies the adapter command and control port. User commands are sent to this port on localhost.

Event Stream Processor Parameters

Event Stream Processor parameters configure communication between Event Stream Processor and the Flex adapter.

These parameters are defined in the `controller.xsd` file in the `config` directory.

Parameter Name	Type	Description
espAuthType	string	<p>(Required) Specifies method used to authenticate to the Event Stream Processor. Valid values are:</p> <ul style="list-style-type: none"> • server_rsa – RSA authentication using keystore • user_password – Kerberos and LDAP authentication • none – No authentication <p>If the adapter is operated as a Studio plug-in, espAuthType is overridden by the Authentication Mode Studio start-up parameter.</p>
espUser	string	<p>(Required) Specifies user name required to log in to Event Stream Processor. It is required for any authentication scheme other than none (see espAuthType). No default value.</p>
espPassword	string	<p>(Required) Specifies the password required to log in to Event Stream Processor. Required for any authentication scheme other than none (see espAuthType).</p> <p>Includes an "encrypted" attribute indicating whether the espPassword value is encrypted. Default value is false. If set to true, the password value is decrypted using espRSAKeyStore and espRSAKeyStorePassword.</p>
espProjectUri	string	<p>(Required) Specifies the total project Uri to connect to Event Stream Processor cluster. For example, <code>esp://localhost:19011/ws1/p1</code>.</p>
pulseInterval	non-negative integer	<p>(Optional) Specifies time interval, in seconds, during which outbound record changes are coalesced by Event Stream Processor, then received by the adapter as a single event.</p> <p>If not set or set to 0, record changes are received individually as they occur.</p>

Parameter Name	Type	Description
espHeartbeatPeriod	positive integer	(Optional) Specifies number, in seconds, that adapter waits before sending the next heartbeat to Event Stream Processor. If Event Stream Processor fails to receive two consecutive heartbeats, all records the adapter publishes are marked stale. Default value is 10.
recordQueueCapacity	positive integer	(Optional) Specifies capacity of the record queues. Default value is 4096.
maxPubPoolSize	positive integer	(Optional) Specifies the maximum size of the record pool. Record pooling allows for faster publication. Default value is 256.
maxPubPoolTime	positive integer	(Optional) Specifies 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	boolean	(Optional) If set to true, pooled messages are published to Event Stream Processor in transactions. If set to false, they are published in envelopes. Default value is false.
espRSAKeyStore	string	(Dependent required) Specifies the location of the RSA keystore, and is used to decrypt the password value. Required if espAuthType is set to <code>server_rsa</code> , or the encrypted attribute for espPassword is set to true, or both.
espRSAKeyStorePassword	string	(Dependent required) Specifies the keystore password, and is used to decrypt the password value. Required if espAuthType is set to <code>server_rsa</code> , or the encrypted attribute for espPassword is set to true, or both.
espEncryptionAlgorithm	string	(Optional) Used when the encrypted attribute for espPassword is set to true. If left blank, RSA is used as default.

Flex Server Settings

The **serverPort** parameter specifies the port on which the adapter runs its Flex server.

This parameter is defined in `flexadapter.xsd` file in the `config` directory.

Parameter Name	Type	Description
serverPort	int	(Required) Specifies the port on which the adapter runs its Flex server.

Sample Flex Configuration File

Sample configuration file (`adapter.xml`) for the Flex adapter.

This file is in the `example` folder.

```
<adapter>
<!-- Adapter Controller -->
<controller>
  <controllerPort>13579</controllerPort>
</controller>

<!-- Sybase ESP Server settings -->
<esp>
  <espConnection>
    <espProjectUri>esp://localhost:19011/w1/p1</espProjectUri>
  </espConnection>

  <espSecurity>
    <espUser>espuser</espUser>
    <espPassword encrypted="false">espuser</espPassword>
    <espAuthType>none</espAuthType>
  <!-- <espRSAKeyStore>/keystore/keystore.jks</espRSAKeyStore>
    <espRSAKeyStorePassword>Sybase123</espRSAKeyStorePassword> --
  >
    <espEncryptionAlgorithm>RSA</espEncryptionAlgorithm>
  </espSecurity>
  <maxPubPoolSize>1</maxPubPoolSize>
</esp>

<!-- Flex specific -->
<flex>
  <serverPort>23456</serverPort>
</flex>
</adapter>
```

Logging

The adapter uses the Apache `log4j` API to log errors, warnings, and information and debugging messages.

The `log4j.properties` file contains the logging configuration. A sample `log4j.properties` file is part of the adapter distribution.

Setting the logging level to `DEBUG` may result in very large log files. The default level is `INFO`.

Refer to <http://logging.apache.org/log4j> for details on logging configuration.

Operation

Start, stop, or check adapter status from the command line.

Starting the Flex Adapter

To start the Flex adapter from the command line, start Event Stream Processor and execute the `start` command.

Prerequisites

Ensure that the port on which the adapter is listening for client connections is open for TCP connections from the machines where the Flex clients are to be run.

Task

1. Start Event Stream Processor.

Windows:

1. Start the example cluster.

```
cd %ESP_HOME%\cluster\nodes\node1
%ESP_HOME%\bin\esp_server --cluster-node node1.xml
```

2. Compile CCL to create CCX.

```
%ESP_HOME%\bin\esp_compiler -i model.ccl -o model.ccx
```

3. Deploy the project on the cluster.

```
%ESP_HOME%\bin\esp_cluster_admin" --uri=esp://localhost:19011
--username=sybase --password=sybase --add_project --workspace-
name=w1 --project-name=p1 --ccx=model.ccx
```

4. Start the deployed project on the cluster.

```
%ESP_HOME%\bin\esp_cluster_admin" --uri=esp://localhost:19011
--username=sybase --password=sybase --start_project --
workspace-name=w1 --project-name=p1
```

UNIX:

1. Start the example cluster.

```
cd $ESP_HOME/cluster/nodes/node1
$ESP_HOME/bin/esp_server --cluster-node node1.xml
```

2. Compile CCL to create CCX.

```
$ESP_HOME/bin/esp_compiler -i model.ccl -o model.ccx
```

3. Deploy the project on the cluster.

```
$ESP_HOME/bin/esp_cluster_admin" --uri=esp://localhost:19011
--username=sybase --password=sybase --add_project --workspace-
name=w1 --project-name=p1 --ccx=model.ccx
```

4. Start the deployed project on the cluster.

```
$ESP_HOME/bin/esp_cluster_admin" --uri=esp://localhost:19011
--username=sybase --password=sybase --start_project --
workspace-name=w1 --project-name=p1
```

2. Start the adapter.

Operating System	Step
UNIX	Open a terminal window and enter: <pre>cd \$ESP_HOME/adapters/flex/bin ./adapter.sh <configuration file path> start</pre>
Windows	Open a command window and enter: <pre>cd %ESP_HOME%/adapters/flex/bin adapter.bat <configuration file path> start</pre>

See also

- *Start Command* on page 176

Checking the Flex Adapter Status

To check the Flex adapter status from the command line, execute the **status** command.

Operating System	Step
UNIX	Open a terminal window and enter: <pre>cd \$ESP_HOME/adapters/flex/bin ./adapter.sh <configuration file path> status</pre>
Windows	Open a command window and enter: <pre>cd %ESP_HOME%/adapters/flex/bin adapter.bat <configuration file path> status</pre>

You see the adapter status, which is either running or stopped.

See also

- *Status Command* on page 176

Stopping the Flex Adapter

To stop the Flex adapter from the command line, execute the **stop** command.

Operating System	Step
UNIX	Open a terminal window and enter: <pre>cd \$ESP_HOME/adapters/flex/bin ./adapter.sh <configuration file path> stop</pre>
Windows	Open a command window and enter: <pre>cd %ESP_HOME%/adapters/flex/bin adapter.bat <configuration file path> stop</pre>

See also

- *Stop Command* on page 176

Example: Sending a Subscription Request

Send a subscription request to the adapter, and see a stream record from Event Stream Processor in a Web browser.

Prerequisites

- Install a Web server (port default is 80), Flash policy server (default is 843), and a Web browser with the Flash plug-in.
- The Web server and the policy server must be running on the same machine on which the adapter is installed.
- Copy the `example.swf` file from the adapter `example` directory to the content area of the Web server.
- If the Web server and policy server are running on a different machine than the one on which the adapter is installed, ensure the ports listed above are open for TCP connections from the machine where the Web browser is running.
- The Flex Server port default is 23456.
- The Web browser can be used on the same machine or on a different machine.

Task

1. Edit the `adapter.sh` script.

2. Set the JAVA_HOME environment variable to the directory where the Java Runtime Environment (JRE) is installed.
3. Start Event Stream Processor.

Operating System	Step
UNIX	Open a terminal window: <ol style="list-style-type: none"> 1. Start the example cluster: <code>start_server_cluster.sh</code> 2. Start the project on the cluster: <code>start_project.sh</code>
Windows	Open a command window: <ol style="list-style-type: none"> 1. Start the example cluster: <code>start_server_cluster.bat</code> 2. Add project to the cluster, and start it on the cluster: <code>start_project.bat</code>

4. Start the adapter.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./adapter.sh</code>
Windows	Open a command window and enter: <code>adapter.bat</code>

5. Wait five to ten seconds for the adapter to initialize.
6. Upload a stream record.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./upload.sh</code>
Windows	Open a command window and enter: <code>upload.bat</code>

7. Point the Web browser to the `example.swf` file. For example:

`http://localhost:80/sybase/example.swf`

8. You see this stream record in the browser window:

```
Stream = Stream1
Opcode = i
Symbol = IBM
Price = 12.50
```

HTTP Output Adapter

Adapter type: httpplugin. The Sybase Event Stream Processor HTTP adapter publishes data from Event Stream Processor to external clients.

The HTTP adapter:

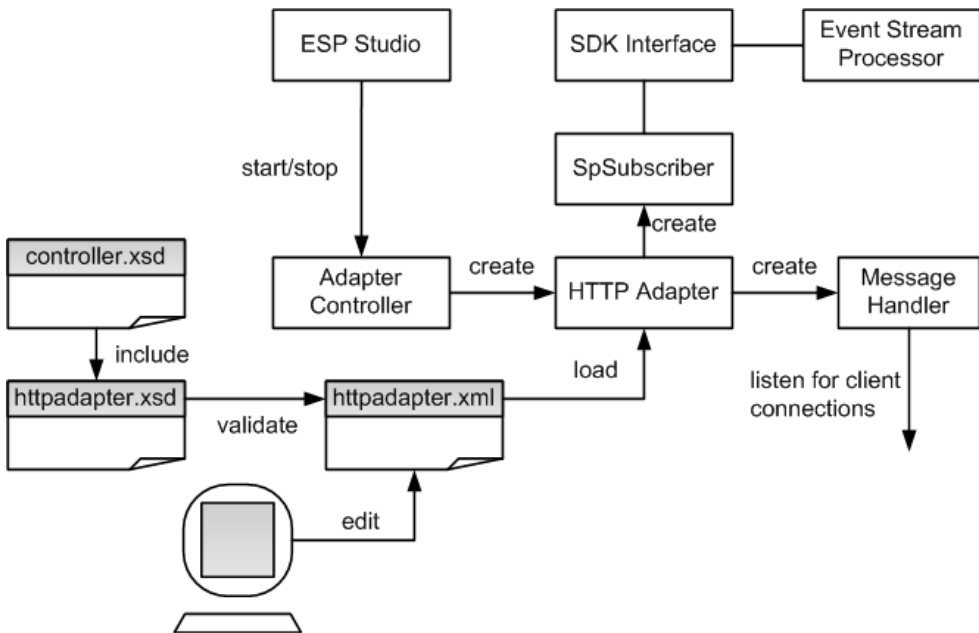
- Runs an internal HTTP server, listens to and accepts client connections
- Extracts SQL queries from client requests and subscribes to streams
- Converts stream records into XML, and sends XML to clients in chunk-coded HTTP responses

Control Flow

The adapter loads its configuration from a file (for example, `adapter.xml`) and validates it against the adapter schema (`httpadapter.xsd`), which includes the API-wide controller schema (`controller.xsd`).

You cannot edit schemas.

Figure 8: HTTP Adapter Control Flow



The Adapter Controller creates an instance of the adapter, receives and executes user commands. The Adapter Controller executes **start**, **stop**, and **status** commands.

Start Command

The **start** command configures and starts the adapter command and control interface, gets the Message Handler to start listening for client connections, and connects the SpSubscriber component to Event Stream Processor via the SDK interface.

The adapter ignores the **start** command if it is executed when there is a running instance of the adapter, and a warning is sent.

See also

- *Starting the HTTP Adapter* on page 198

Stop Command

The **stop** command disconnects the SpSubscriber from Event Stream Processor, causes the Message Handler to finalize the HTTP responses to the existing clients, disconnect them and stop listening for connections from new clients, and terminates the adapter process.

If the **stop** command is executed when there is no instance of a running adapter, the command is ignored and a warning is sent.

See also

- *Stopping the HTTP Adapter* on page 199

Status Command

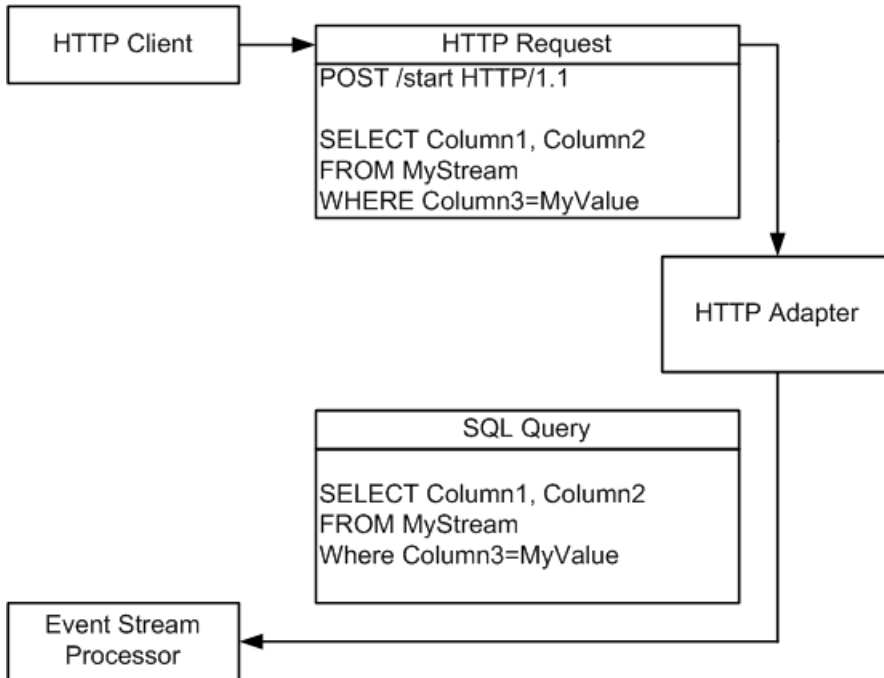
The **status** command reports the adapter status, and the Adapter Controller prints out its status: either running or stopped.

See also

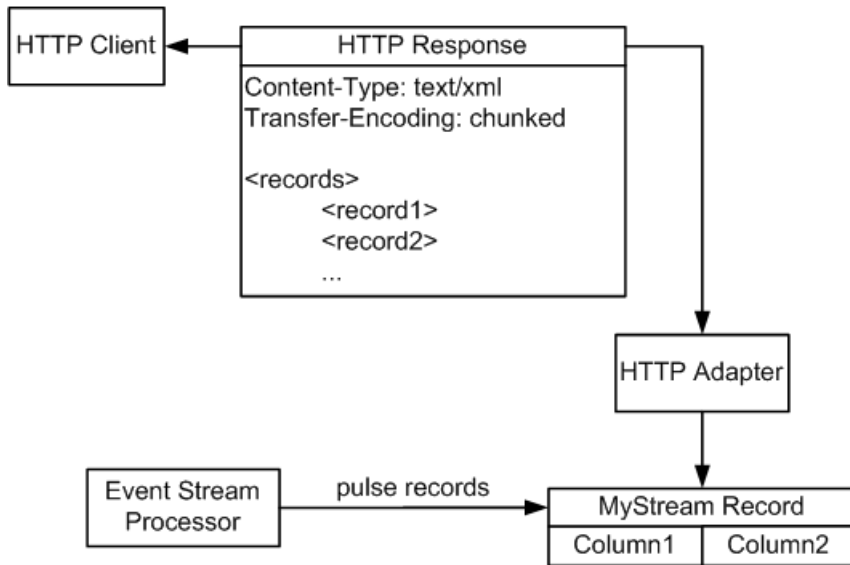
- *Checking the HTTP Adapter Status* on page 199

Message Flow

The message flow between the adapter and an HTTP client is initiated when the client sends a POST request with the **start** command in it and a body consisting of a SQL query to Event Stream Processor.



Changes in the corresponding stream are pulsed back to the HTTP client as XML-formatted chunk-coded HTTP responses.



The pulse interval is specified in the adapter configuration. In the event of a failover, the SDK API switches, as configured, to the spare Event Stream Processor instance without message loss.

See also

- *Event Stream Processor Parameters* on page 193

Setting the JAVA_HOME Environment Variable

Set the JAVA_HOME environment variable to point to the Java directory.

Prerequisites

Install Java Runtime Environment version 1.6.0_26 or higher.

Task

Set the JAVA_HOME environment variable to the directory path where Java Runtime Environment 1.6.0_26 or higher is installed.

Configuration

Configuration information for the HTTP adapter.

HTTP Adapter Directory

The adapter directory contains all files, such as configuration files, templates, examples, and JAR files, relating to the adapter.

```
README.txt Quick Guide
ReleaseNotes.txt Release Notes
```

CHAPTER 2: Adapters Supported by Event Stream Processor

```
bin/  
adapter.bat Standalone adapter startup script  
adapter.sh Standalone adapter startup script  
adapter-plugin.bat Plug-in connector startup script  
adapter-plugin.sh Plug-in connector startup script  
  
config/  
controller.xsd Controller schema  
log4j.properties Sample logging configuration  
httpadapter.xsd Adapter schema  
login.config Authentication configuration  
  
example/ Working example  
  
lib/  
esp_http_adapter.jar http adapter library  
  
javadoc/  
  
adapterapi/ Adapter API Javadoc  
httpadapter/ Http Adapter Javadoc  
  
Common jars are located:  
  
$ESP_HOME/adapters/jar  
  
activation.jar Java mail library  
adapterapi.jar Adapter API code  
axis.jar Webservices jar  
commons-codec-1.3.jar Required by SDK API  
commons-discovery-0.2.jar  
commons-logging-1.1.jar Logging library  
esp_java_sdk-0.2.jar ESP SDK library  
jaxrpc-api-1.1.jar Required by ESP SDK  
log4j-1.2.14.jar Logging library  
mail.jar Java mail library  
saaj-api-1.3.jar Webservices jar  
ws-commons-util-1.0.2.jar Required by ESP SDK  
wsdl4j-1.5.1.jar  
xercesImpl.jar XML parser library  
xmlrpc-client-3.1.3.jar Required by ESP SDK  
xmlrpc-common-3.1.3.jar Required by ESP SDK  
xmlrpc-server-3.1.3.jar Required by ESP SDK
```

Schema and Configuration File

The adapter configuration is loaded from a file and validated against the adapter schema.

The `example` folder contains a sample adapter configuration file. Provide a valid configuration file, and ensure the adapter configuration validates against the adapter schema.

Adapter Controller Parameter

The **controllerPort** parameter specifies the adapter command and control port.

Parameter Name	Type	Description
controllerPort	positive integer	(Required) Specifies the adapter command and control port. User commands are sent to this port on localhost.

Event Stream Processor Parameters

Event Stream Processor parameters configure communication between Event Stream Processor and the HTTP adapter.

These parameters are defined in the `controller.xsd` file in the `config` directory.

Parameter Name	Type	Description
espAuthType	string	(Required) Specifies method used to authenticate to the Event Stream Processor. Valid values are: <ul style="list-style-type: none"> • server_rsa – RSA authentication using keystore • user_password – Kerberos and LDAP authentication • none – No authentication If the adapter is operated as a Studio plug-in, espAuthType is overridden by the Authentication Mode Studio start-up parameter.
espUser	string	(Required) Specifies user name required to log in to Event Stream Processor. It is required for any authentication scheme other than none (see espAuthType). No default value.
espPassword	string	(Required) Specifies the password required to log in to Event Stream Processor. Required for any authentication scheme other than none (see espAuthType). <p>Includes an "encrypted" attribute indicating whether the espPassword value is encrypted. Default value is false. If set to true, the password value is decrypted using espRSAKeystore and espRSAKeystorePassword.</p>

Parameter Name	Type	Description
espProjectUri	string	(Required) Specifies the total project Uri to connect to Event Stream Processor cluster. For example, <code>esp://localhost:19011/ws1/p1</code> .
pulseInterval	non-negative integer	(Optional) Specifies time interval, in seconds, during which outbound record changes are coalesced by Event Stream Processor, then received by the adapter as a single event. If not set or set to 0, record changes are received individually as they occur.
espHeartbeatPeriod	positive integer	(Optional) Specifies number, in seconds, that adapter waits before sending the next heartbeat to Event Stream Processor. If Event Stream Processor fails to receive two consecutive heartbeats, all records the adapter publishes are marked stale. Default value is 10.
recordQueueCapacity	positive integer	(Optional) Specifies capacity of the record queues. Default value is 4096.
maxPubPoolSize	positive integer	(Optional) Specifies the maximum size of the record pool. Record pooling allows for faster publication. Default value is 256.
maxPubPoolTime	positive integer	(Optional) Specifies 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	boolean	(Optional) If set to true, pooled messages are published to Event Stream Processor in transactions. If set to false, they are published in envelopes. Default value is false.

Parameter Name	Type	Description
espRSAKeyStore	string	(Dependent required) Specifies the location of the RSA keystore, and is used to decrypt the password value. Required if espAuthType is set to <code>server_rsa</code> , or the encrypted attribute for espPassword is set to true, or both.
espRSAKeyStorePassword	string	(Dependent required) Specifies the keystore password, and is used to decrypt the password value. Required if espAuthType is set to <code>server_rsa</code> , or the encrypted attribute for espPassword is set to true, or both.
espEncryptionAlgorithm	string	(Optional) Used when the encrypted attribute for espPassword is set to true. If left blank, RSA is used as default.

See also

- *Message Flow* on page 190

HTTP Server Settings

The **httpPort** and **contentType** parameters specify HTTP Server settings.

Parameter	Type	Description
httpPort	integer	(Required) Specifies the port on which the adapter runs its HTTP server.
contentType	string	(Required) Specifies the content type of HTTP responses. The adapter supports the <code>text/plain</code> and <code>text/html</code> content types.

Sample HTTP Configuration File

Sample configuration file (`adapter.xml`) for the HTTP adapter.

This file is in the `example` folder.

```
<adapter>
- <!-- Adapter Controller
  -->
- <controller>
  <controllerPort>13579</controllerPort>
</controller>
- <!-- Event Stream Processor Settings
  -->
- <esp>
  <espConnection>
  <espHost>localhost</espHost>
```

```

    <espPort>22000</espPort>
- <!--      <espProjectUri>esp://localhost:19011/wsl/pl</
espProjectUri>
-->
  </espConnection>
- <espSecurity>
  <espUser>espuser</espUser>
  <espPassword encrypted="false">espuser</espPassword>
  <espAuthType>none</espAuthType>
- <!--
    <espRSAKeyFile>/keyfilepath/espuser.private.der</espRSAKeyFile>
    <espRSAKeyStore>/keystore/keystore.jks</espRSAKeyStore>
    <espRSAKeyStorePassword>Sybase123</espRSAKeyStorePassword>

-->
  <espEncryptionAlgorithm>RSA</espEncryptionAlgorithm>
</espSecurity>
<maxPubPoolSize>1</maxPubPoolSize>
</esp>
- <!-- HTTP specific
-->
- <http>
  <httpPort>23456</httpPort>
  <contentType>text/html</contentType>
</http>
</adapter>

```

HTTP Output Adapter

The HTTP Output adapter receives SQL queries wrapped in HTTP requests from a client application, such as a Web browser, and sends chunk-coded stream content back to the client.

You can configure the adapter on any source stream as an outbound data location. The authentication method is set to Event Stream Processing standards: none, rsa, or gssapi. To use this adapter, ensure Sybase HTTP adapter version 1.0 or later is installed.

Property Label	Property Id	Type	Description
Connector Directory Path	baseDir	directory	(Required) Specifies the absolute path to the adapter installation directory. This property is ignored if the Connector Remote Directory Path property is supplied. No default value.
Configuration File Path	configFilePath	configFilename	(Required) Specifies the absolute path to the adapter configuration file. This property is ignored if the Remote Configuration Path property is supplied. No default value.

Property Label	Property Id	Type	Description
Connector Remote Directory Path	remoteBaseDir	string	(Advanced) Specifies the path to the adapter remote base directory, for remote execution only. If this property is supplied, the Connector Directory Path is ignored. No default value.
Remote Configuration File Path	remoteConfigFilePath	string	(Advanced) Specifies the path to the adapter remote configuration file, for remote execution only. If this property is supplied, the Configuration File Path property is ignored. No default value.
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.

Logging

The adapter uses the Apache `log4j` API to log errors, warnings, and information and debugging messages.

The `log4j.properties` file contains the logging configuration. A sample `log4j.properties` file is part of the adapter distribution.

Set the desired logging level in the `log4j.properties` file. Setting the logging level to `DEBUG` may result in very large log files. The default level is `INFO`. Raw IDC messages are logged at the `DEBUG` level. Refer to <http://logging.apache.org/log4j> for details on logging configuration.

Operation

You can operate the HTTP Output adapter from the command line.

Starting the HTTP Adapter

To start the HTTP adapter from the command line, start Event Stream Processor and execute the **start** command.

1. Start Event Stream Processor.

Windows:

1. Start the example cluster.

```
cd %ESP_HOME%\cluster\nodes\node1
%ESP_HOME%\bin\esp_server --cluster-node node1.xml
```

2. Compile CCL to create CCX.

```
%ESP_HOME%\bin\esp_compiler -i model.ccl -o model.ccx
```

3. Deploy the project on the cluster.

```
%ESP_HOME%\bin\esp_cluster_admin" --uri=esp://localhost:19011
--username=sybase --password=sybase --add_project --workspace-
name=w1 --project-name=p1 --ccx=model.ccx
```

4. Start the deployed project on the cluster.

```
%ESP_HOME%\bin\esp_cluster_admin" --uri=esp://localhost:19011
--username=sybase --password=sybase --start_project --
workspace-name=w1 --project-name=p1
```

UNIX:

1. Start the example cluster.

```
cd $ESP_HOME/cluster/nodes/node1
$ESP_HOME/bin/esp_server --cluster-node node1.xml
```

2. Compile CCL to create CCX.

```
$ESP_HOME/bin/esp_compiler -i model.ccl -o model.ccx
```

3. Deploy the project on the cluster.

```
$ESP_HOME/bin/esp_cluster_admin" --uri=esp://localhost:19011
--username=sybase --password=sybase --add_project --workspace-
name=w1 --project-name=p1 --ccx=model.ccx
```

4. Start the deployed project on the cluster.

```
$ESP_HOME/bin/esp_cluster_admin" --uri=esp://localhost:19011
--username=sybase --password=sybase --start_project --
workspace-name=w1 --project-name=p1
```

2. Start the adapter.

Operating System	Step
UNIX	Open a terminal window and enter: <pre>cd \$ESP_HOME/adapters/http/bin ./adapter.sh <configuration file path> start</pre>
Windows	Open a command window and enter: <pre>cd %ESP_HOME%/adapters/http/bin adapter.bat <configuration file path> start</pre>

See also

- *Start Command* on page 189

Checking the HTTP Adapter Status

To check the HTTP adapter status from the command line, execute the **status** command.

Operating System	Step
UNIX	Open a terminal window and enter: <pre>cd \$ESP_HOME/adapters/http/bin ./adapter.sh <configuration file path> status</pre>
Windows	Open a command window and enter: <pre>cd %ESP_HOME%/adapters/http/bin adapter.bat <configuration file path> status</pre>

You see the adapter status, which is either running or stopped.

See also

- *Status Command* on page 189

Stopping the HTTP Adapter

To stop the HTTP adapter from the command line, execute the **stop** command.

Operating system	Step
UNIX	Open a terminal window and enter: <pre>cd \$ESP_HOME/adapters/http/bin ./adapter.sh <configuration file path> stop</pre>
Windows	Open a command window and enter: <pre>cd %ESP_HOME%/adapters/http/bin adapter.bat <configuration file path> stop</pre>

See also

- *Stop Command* on page 189

Example: Sending, Receiving, and Viewing Data

Use the working example provided in the adapter distribution to learn how to send a SQL query to the adapter, and receive XML-formatted stream data from Event Stream Processor and view it in a Web browser.

Prerequisites

You have a network connection.

Task

1. Start Event Stream Processor.

Operating System	Step
UNIX	Open a terminal window: <ol style="list-style-type: none"> 1. Start the example cluster: <code>start_server_cluster.sh</code> 2. Start the project on the cluster: <code>start_project.sh</code>
Windows	Open a command window: <ol style="list-style-type: none"> 1. Start the example cluster: <code>start_server_cluster.bat</code> 2. Add project to the cluster, and start it on the cluster: <code>start_project.bat</code>

2. Edit the `adapter.sh` script.
3. Set the `JAVA_HOME` environment variable to the directory where the Java Runtime Environment (JRE) is installed.
4. Start the adapter.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./adapter.sh</code>
Windows	Open a command window and enter: <code>adapter.bat</code>

- Wait five to ten seconds for the adapter to initialize.
- Start uploading stream records.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./upload.sh</code>
Windows	Open a command window and enter: <code>upload.bat</code>

- Load the HTTPAdapterClient.html page in a Web browser.
- Enter a valid SQL query, for example:

```
SELECT * FROM Stream1
```

Note: Queries that return a smaller number of records may not appear on some browsers because those browsers appear to expect a certain amount of data to be present in the buffer cache before they display the data. For example:

```
SELECT * FROM Stream1
where intcol = 10
```

- Mozilla FireFox Browser: displays record.
- Google Chrome Browser: does not display record.
- Internet Explorer Browser: does not display record.

```
SELECT * FROM Stream1
where intCol > 10
```

(executed in debug mode)

- Mozilla FireFox Browser: records appear after they are sent to the browser.
- Google Chrome Browser: records appear after the fifth record is sent to the browser.
- Internet Explorer Browser: records appear after the 20th record is sent to the browser.

- Click **Submit**.
- Note the records being streamed into the Web browser window.

KDB Adapter

Adapter types: KDBInput, KDBOutput. The Sybase Event Stream Processor KDB Input and Output adapters allow data to be loaded from a kdb+ database into an Event Stream Processor project, and for output from an Event Stream Processor project to be stored in a kdb+ database.

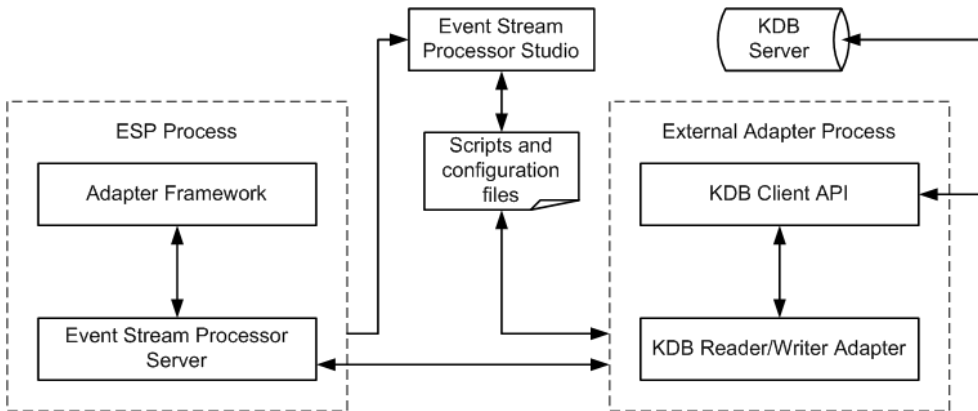
Note: The KDB Input and Output adapters do not support the Solaris AMD platform.

Control Flow

KDB Input and Output adapters are operated by a set of script files.

The adapter scripts use these basic commands:

Figure 9: KDB Adapter Control Flow



Start Command

The **start** command starts the KDB Input or Output adapter, the KDB client API, and the KDB database server, and connects them to Event Stream Processor via the SDK interface.

Stop Command

The **stop** command stops the KDB Input or Output adapter, and closes the connection between the KDB database server and Event Stream Processor.

Datatype Mapping for the KDB Adapter

Event Stream Processor datatypes map to KDB datatypes, and KDB datatypes map to Event Stream Processor datatypes.

KDB Datatypes to ESP Datatypes

KDB datatypes map to Event Stream Processor datatypes.

KDB Datatype	Character	ESP Datatype
boolean	b	boolean
byte	x	integer
short	h	integer
int	i	integer
long	j	long
real	e	float
float	f	float
symbol	s	string
date	d	date
datetime	z	timestamp
time	t	timestamp
month	m	date
minute	u	interval
second	v	interval

ESP Datatypes to KDB Datatypes

Event Stream Processor datatypes map to KDB datatypes.

ESP Datatype	Character	KDB Datatype
integer	i	int
long	j	long
float	f	float
money	f	float
string	s	symbol
date	z	datetime

ESP Datatype	Character	KDB Datatype
timestamp	z	datetime
bigdatetime	j	long
interval	j	long
boolean	b	boolean

KDB Input Adapter

The KDB Input adapter reads kdb or kdb+tick database tables.

By default, the adapter matches the field names (in a case-insensitive manner) to determine the mapping between the source kdb+tick table and the target stream. The KDB Output adapter also supports custom field-mapping.

This adapter supports schema discovery.

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is KDBInput.

Property Label	Property ID	Type	Description
KDB Server	server	string	(Required) Specifies the name or IP address of the database server machine. Default value is localhost.
KDB Port	port	range	(Required) Specifies the IP port of the database listener. Default value is 5001. Minimum value is 0, maximum value is 65535.
Event Stream Processor User ID	espUser	string	(Optional) Specifies the user name for connecting to the Event Stream Processor. Default value is t.
Event Stream Processor Password	espPassword	password	(Optional) Specifies the password for the Event Stream Processor user ID. Note: If RSA authentication is used, this property can be left blank. Default value is t.

Property Label	Property ID	Type	Description
Authentication	authentication	choice	<p>(Optional) Specifies the method used to authenticate to the kdb or kdb+tick database tables. Valid values are:</p> <ul style="list-style-type: none"> • value="none" label="None" • value="pam" label="PAM" • value="rsa" label="RSA" • value="gssapi" label="Kerberos V5" <p>No default value.</p>
Project URI	projectUri	string	(Optional) Specifies the URI to connect to a project in cluster environment. No default value.
RSA Key File	rsaKeyFile	filename	(Optional) Specifies the RSA private-key file name and location. No default value.
KDB User	user	string	(Optional) Specifies the user ID for the database connection. No default value.
KDB Password	password	password	(Optional) Specifies the password for the database connection. No default value.

Property Label	Property ID	Type	Description
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.
Source Table	table	tables	(Advanced) Specifies the name of the source table to retrieve data from. <hr/> Note: This property supports SQL query statements. <hr/> No default value.
Field Mapping	permutation	string	(Advanced) Specifies the mapping between the in-Event Stream Processor and external fields. Format is: ESPCol- um1=KDBCol- um1:ESPCol- um2=KDBCol- um2 . . . No default value.

Property Label	Property ID	Type	Description
Streaming Mode	mode	choice	<p>(Advanced) Specifies if the adapter should connect to a kdb+tick database and read in streaming data or execute the supplied query and feed the result to Event Stream Processor. Valid value list:</p> <ul style="list-style-type: none"> • value="stream", label="Stream" • value="pull", label="One time pull" <p>Default value is "stream".</p>
Polling Interval	pollInterval	int	<p>(Advanced) Specifies the number of seconds to wait before re-executing query in pull mode. If set to 0, the query is not reexecuted. Default value is 0.</p>
Block Size	blockSize	int	<p>(Advanced) Specifies the number of records to block into one pseudo-transaction. Default value is 64.</p>
Async Mode	async	boolean	<p>(Advanced) If set to true, the adapter does not wait for an acknowledgement from Event Stream Processor that is received the data. Default value is false.</p>
Encrypt Connection	encrypt	boolean	<p>(Advanced) If set to true, the traffic between Event Stream Processor and the adapter is encrypted. Default value is false.</p>

Property Label	Property ID	Type	Description
Print Debug Info	debug	boolean	(Advanced) Specifies if the adapter prints additional debugging information to the console. Default value is false.
Use Transaction Blocks	useTransaction	boolean	(Advanced) For better performance, use transaction blocks instead of envelopes while sending data to Event Stream Processor. Default value is false.
Connection Retries	connectionRetries	int	(Advanced) Specifies the number of times to retry connection if it breaks. Default value is 1.
Event Stream Processor Host Name	gatewayHost	string	(Advanced) Specifies the explicit gateway host name. No default value.
Parameter File	x_paramFile	filename	(Advanced) Specifies the file to write the parameters into, to pass to the external process. No default value.
Parameter File Format	x_paramFormat	choice	(Advanced) Specifies the format in which the external process expects the parameters. Valid value list: <ul style="list-style-type: none"> • value = prop, label = "Java properties" • value= shell, label="Unix shell assignments" • value = xml, label = "Simple XML" Default value is "prop".

Known limitations:

- If the kdb+tick databases are not running when the adapter tries to make a connection, the adapter waits indefinitely until the kdb+tick database is started.

Note: This issue occurs only if the kdb+tick database and Event Stream Processor are running on different machines.

- If the connection to the database is broken, any updates that happen between the time the connection is broken and reestablished are lost.

KDB Output Adapter

The KDB Output adapter publishes stream data from Event Stream Processor to a KDB+tick database table.

By default, the adapter matches the field names (in a case-insensitive manner) to decide the mapping between the source KDB+tick table and the target stream. The KDB Output adapter supports custom field-mapping.

If you use the CCL **ATTACH ADAPTER** statement to attach an adapter, you must supply the adapter type. The type for this adapter is `KDBOutput`.

Property Label	Property ID	Type	Description
KDB Server	server	string	(Required) Name or IP address of the database server machine. No default value.
KDB Port	port	range	(Required) IP port of the database listener. Default value is 5001. Minimum value is 0, maximum value is 65535.
Event Stream Processor User ID	espUser	string	(Optional) User name for connecting to the Event Stream Processor. No default value.
Event Stream Processor Password	espPassword	password	(Optional) Password for the Event Stream Processor user ID. Can be empty if using RSA authentication. No default value.

Property Label	Property ID	Type	Description
Authentication	authentication	choice	<p>(Optional) Authentication mechanism to use. Valid values are:</p> <ul style="list-style-type: none"> • value="none" label="None" • Value value="rsa" label="RSA" • Value value="gssapi" label="Kerberos V5" <p>No default value.</p>
Project URI	projectUri	string	(Optional) Specifies the URI to connect to a project in cluster environment. No default value.
RSA Key File	rsaKeyFile	filename	(Optional) RSA private-key file name and location. No default value.
KDB User	user	string	(Optional) User ID for the database connection. No default value.
KDB Password	password	password	(Optional) Password for the database connection. No default value.

Property Label	Property ID	Type	Description
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.
Target Table	table	tables	(Advanced) Name of the target table to update. No default value.
SQL Query	query	string	(Advanced) The SQL query to filter incoming data. No default value.
Field Mapping	permutation	string	(Advanced) Mapping between the in-Event Stream Processor and external fields. Format is: <code>ESPCol-umn1=KDBCColumn1:ESPCol-umn2=KDBCColumn2...</code> " No default value.

Property Label	Property ID	Type	Description
Streaming Mode	mode	choice	<p>(Advanced) Streaming mode. Valid values are:</p> <ul style="list-style-type: none"> value="stream", label="user.update" value="push", label="use update" <p>No default value.</p>
Async Mode	async	boolean	(Advanced) If set to true, the adapter does not wait for an acknowledgement from Event Stream Processor that is received the data. Default value is false.
Connection Retries	connectionRetries	int	(Advanced) Number of times to retry connection if it breaks. Default value is 1.
KDB Batch size	blockSize	int	(Advanced) Maximum number of records in a single KDB write batch. Default value is 5000.
BASE Data Only	baseOnly	boolean	(Advanced) If set to true, does not retrieve data existing at connection time. Default value is false.
Lossy Subscriber	lossy	boolean	(Advanced) If set to true, the Event Stream Processor drops records if connection slows down. Default value is false.

Property Label	Property ID	Type	Description
Pulse Interval	pulsed	int	(Advanced) If set as a non-zero value, the subscription is created in pulsed mode with the specified period. Default value is 0.
Shine Through	shine	boolean	(Advanced) If set to true, subscribe send data with shine through. Default value is false.
Droppable Subscription	droppable	boolean	(Advanced) If set to true, Event Stream Processor drops the subscription if data is being backed up. Default value is false.
Preserve Transaction Blocks	originalBlocks	boolean	(Advanced) If set to true, Event Stream Processor preserves transaction boundaries in subscribed data. Default value is false.
Debug	debug	boolean	(Advanced) If set to true, a debug message of connection between KDB database server and Event Stream Processor is shown together with other Event Stream Processor logs. Default value is false.
Omitted Fields	omitFields	string	(Advanced) Comma-delimited list of KDB fields to omit. <hr/> Note: Omitted fields are not sent. <hr/> No default value.
Ignored Fields	ignoreFields	string	(Advanced) Comma-delimited list of KDB fields to ignore, these are set to NULL. No default value.

Property Label	Property ID	Type	Description
Quiet Mode	quietMode	boolean	(Advanced) If true, a KDB log message is not show in standard error output. Default value is false.
Encrypt Connection	encrypt	boolean	(Advanced) If set to true, the traffic between Event Stream Processor and the KDB adapter is encrypted. Default value is false.
Parameter File	x_paramFile	filename	(Advanced) File to write the parameters into, to pass to the external process. Default value is <temp>\PARAMETER_FILE.txt.
Parameter File Format	x_paramFormat	choice	(Advanced) Format in which the external process expects the parameters. Valid values are: <ul style="list-style-type: none"> • value = prop, label = "Java properties" • value = shell, label= "Unix shell assignments" • value = xml, label = "Simple XML" Default value is "prop".

Log File Input Adapter

The Sybase Event Stream Processor Log File Input adapter reads from a log file and sends data to a stream.

For each log record read from the file, the adapter sends one row (message) to the stream. The log file can be in any of the following formats:

- Common log format

- Combined log format
- Extended common log format

The choice of formats allows you to read log files from Apache, Tomcat, IIS, and other datasources that write their log files in one of these formats. You can customize the appropriate `.properties` file to read the log files in other formats.

This adapter is written in Java and uses JavaBeans. Sybase assumes, if you use this adapter, that you have extensive knowledge and experience with JavaBeans and `.properties` files.

You can use this adapter to read either live or historical log files. Historical files are complete and can be read once from beginning to end. Live files are those to which data continues to be added.

There are two types of live files, rotating and advancing. For rotating files, when the log file is full, the program that writes to the file renames the existing file and starts a new file with the original name. Typically the new file name is based on the original file name with a suffix appended, for example, if the original file name is "access.log", the renamed files are "access.log.1", "access.log.2", and so on. The Log File Input adapter always reads the original file name. The adapter starts over at the beginning when the old file has been renamed and a new one created.

For advancing files, when one file is full, the log file writer creates a new file and starts writing to that new file. The naming convention is typically a base name plus a suffix, where the suffix might be based on date/time or a sequential number (for example, "access-log.2007-01-01", "access-log.2007-01-02", and so on, or "access.log.1", "access.log.2", and so on.) Regardless of the naming convention, the adapter opens these log files in chronological order by last modification date.

Configuration

Configure your Log File Input adapter by setting values in the `.properties` file for the adapter.

For example, specify the name of the log file to read by setting the **Input.FileName** property. An example `.properties` file is included in the product.

You can use this adapter to read either live or historical log files.

- To read a historical file, specify the file name in the **Input.FileName** property, and set the **Input.WaitForGrowth** property to false.
- To read a live file, whether rotated or advancing, set the **Input.WaitForGrowth** property to true. The Log File Input adapter goes to the end of the file and then reads as new data is appended to the file. When the file size shrinks to zero (after the old log file is renamed and a new, empty one is created), the Log File Input adapter continues reading from the beginning of the new file.

Properties

Lists and briefly describes some of the crucial properties in the `.properties` file. See the example `.properties` file for a complete, up-to-date list of configurable properties for the adapter.

The example `.properties` file contains commented-out examples of the column lists for the common log format and the combined log format. (See the "Parse.Class" section of the file.) To use the Extended Common log format, add column names and datatypes to the `Parse.Format.Common.Columns` property. Since the log file reader is extensible and modifiable, you can define your own formats, and modify the existing formats to match your configurations. For example, if your Apache server has changed the format it writes for common log format or combined log format, you can modify the properties file to match. You can either modify existing entries or create new entries. For example, you can create a new format named "MyUncommonFormat" and define the columns for that format.

Property Name	Type	Description
Input.FileName	string	Lists the name of the log file to read from.
Input.WaitForGrowth	string(true, false, a number of milliseconds, or a number of milliseconds followed by a comma and a number of nanoseconds)	If set to false, the adapter reads until it reaches the end of the file and then exits. If set to true, the adapter continues reading from the log file indefinitely at intervals of 100 milliseconds (the default interval). A value in the format milliseconds or nanoseconds tells the server to continue reading from the log file indefinitely at the specified interval, for example, 10 tells the adapter to read every 10 milliseconds and 0,500000 tells the adapter to check every 500000 nanoseconds, or 2000 times per second. The actual interval between reads is only approximately the interval you specify; the exact interval depends upon your system's hardware and operating system and load.
Parse.Class	string	Specifies the JavaBean used to parse columns in the log file.
Parse.FormatName	string	Specifies the name of the format of the data in the log file. This may be the name of one of the predefined formats, such as common or combined, or it may be the name of a custom format.
Parse.{Format-Name}.Columns (where {Format-Name} is replaced with the name provided by Parse.FormatName)	string	Contains the names and datatypes of the column names in the log file. You may need to customize this property.

Property Name	Type	Description
Parse.{Format-Name}.DateColumn (where {Format-Name} is replaced with the name provided by Parse.FormatName)	string	Contains the name of the date column in the log file. This value must match the actual name of your date column.
Parse.{Format-Name}.TimeColumn (where {FormatName} is replaced with the name provided by Parse.FormatName)	string	(Optional) Contains the name of the time column in the log file. This column is needed only if the date column does not include the time.
Output.Uri	string	URI of the stream to send the rows to.
Output.Columns	string	The string should contain a list of column names separated by spaces. This allows you to match the output to the schema of the stream that you are writing to.
Admin.Input	string	If Admin.Input is set to stdin, then the log file adapter reads stdin looking for administrative commands. The only administrative command it currently supports is exit, which can be requested by any of the input lines: exit, quit, x, or q. The primary intent of this is as a debugging aid. The default is for no administrative request bean.
Output.AuthType	string	(Optional) Specifies authentication type. Valid values are: None, UserPassword, or ServerRSA. Default value is None.
Output.username	string	(Dependent required) Required for all authentication methods other than None. For UserPassword, this specifies the username for the server. For ServerRSA authentication method, this specifies the keystore alias.
Output.password	string	(Dependant required) Required by UserPassword authentication method. Specifies password for the Server.
Output.passwordEncrypted	string	(Dependant required) Specifies whether password in Output.password property is encrypted. Valid values are true or false. Default value is false.
Output.RSAKeystore	string	(Dependent required) Specifies location of keystore file. Required by ServerRSA authentication method or UserPassword (if password is encrypted).

Property Name	Type	Description
Output.RSAKeyStoreAlias	string	(Dependent required) Specifies the keystore alias. Required by UserPassword authentication method if password is encrypted.
Output.RSAKeyStorePassword	string	(Dependent required) Specifies password for the RSA keystore. Required by ServerRSA authentication method or UserPassword (if password is encrypted).
Parse.BigDateTime.Format	string	Specifies format of bigdatetime column. Uses java.text.SimpleDateFormat, which does not support microseconds. To display microseconds, ensure the string UUUUUU is present. Note: If month comes after microseconds, and the number of characters in the month are variable, the data is not parsed correctly. For example, if the full month name is used, September has more characters than July, and July has more than May.

Note: Each input stream has a property (see the stream's Properties tab in Studio) that can specify whether to use the current server timestamp value instead of the row timestamp set by the adapter. If this stream property is set to true, it overrides any row timestamp set by the adapter.

Starting the Adapter from the Command Line

To start the Log File Input adapter from the command line, set the CLASSPATH environment variable and execute the **start** command.

To set the CLASSPATH environment variable, use `createClasspath.sh` (UNIX, Linux) or `createClasspath.bat` (Windows).

Most UNIX-like operating systems place startup scripts in `/etc/init.d`. To start the adapter on such a system, Sybase recommends that you copy the script `logfile_input.rc` to the `/etc/init.d` directory and make any necessary edits.

Note: The `logfile_input.rc` script is a sample, and only works on Linux. To use this on another platform, you need to customize this script.

On systems without `/etc/init.d`, you can use `logfile_input.rc` as the basis for writing your own script to start the Log File Input adapter.

The commands that you can execute through the `logfile_input.rc` file are:

- **Start**
- **Stop**
- **Status**

- **Restart**

Attention: The `logfile_input.rc` script requires you to have write permissions to `/etc/sysconfig`, `/var/run`, and `/var/lock/sybsys`.

1. Run the command **source createClasspath.bat** for Windows or **source createClasspath.sh** for UNIX to set the CLASSPATH environment variable.
2. To run the adapter from the command line:

```
java -cp $CP -Dproperties=FILE.PROPERTIES  
com.sybase.esp.adapters.logFileInput.Main
```

There is no space between the "D" and the word "properties".

NYSE Technologies Input Adapter

Adapter type: `wombatplugin`. The Sybase Event Stream Processor adapter for NYSE Technologies MAMA (NYSE adapter) is used to connect to Wombat market data infrastructure.

The NYSE adapter does not support the Solaris SPARC platform.

The NYSE adapter:

- Connects to the Wombat data feed, opens sessions, and creates and signs off subscriptions
- Translates Wombat messages into Event Stream Processor records

The NYSE adapter requires a separately purchased license. This license does not support the standard SySAM grace period, meaning it cannot run without a valid license.

If you purchased your product from Sybase or an authorized Sybase reseller, go to the secure Sybase Product Download Center (SPDC) at <https://sybase.subscribenet.com> and log in to generate license keys. The license generation process may vary slightly, depending on whether you ordered directly from Sybase or from a Sybase reseller.

If you ordered your product under an SAP contract and were directed to download from SAP Service Marketplace (SMP), you can use SMP at <http://service.sap.com/licensekeys> to generate license keys for Sybase products that use SySAM 2-based licenses.

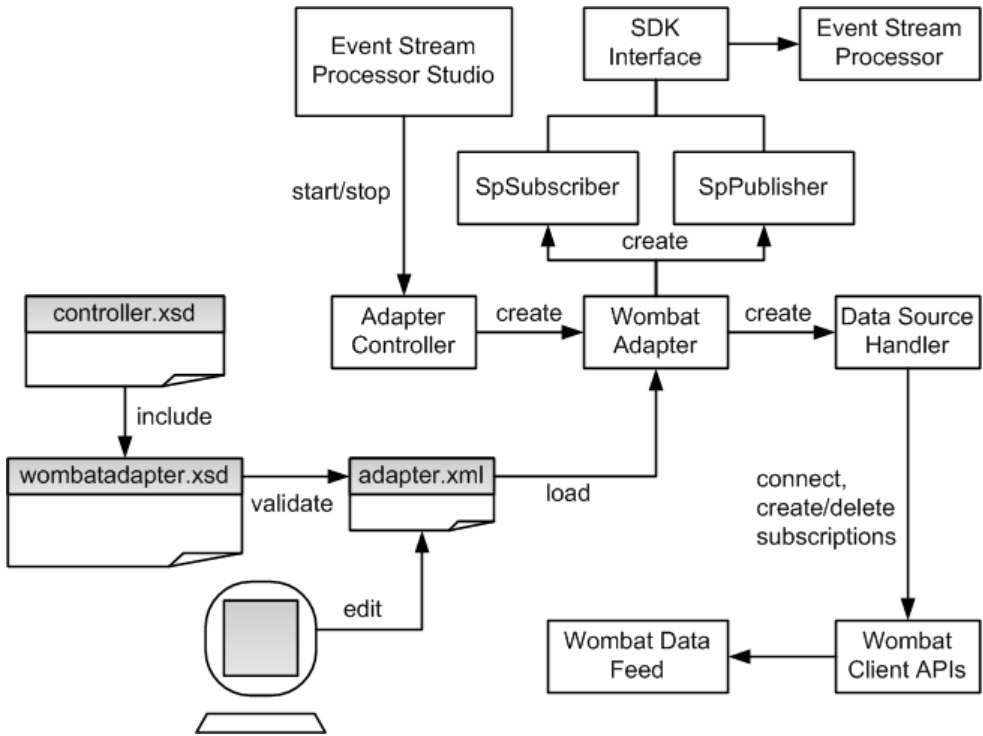
Control Flow

The adapter loads its configuration from a file (for example `adapter.xml`) and validates it against the adapter schema (`wombatadapter.xsd`), which includes the API-wide controller schema (`controller.xsd`).

You cannot edit schemas.

The Adapter Controller creates an instance of the adapters, and receives and executes **start**, **stop**, and **status** commands.

Figure 10: NYSE Technologies Adapter Control Flow



Start Command

The **start** command configures and starts the adapter command and control interface.

The Data Source Handler, which implements the NYSE MAMA and MAMDA Client APIs, connects to the NYSE data feed, initiates a session with it, and downloads the data dictionary. The SpSubscriber and SpPublisher components connect to Event Stream Processor via the SDK interface. SpSubscriber starts listening to the watchlists and SpPublisher is ready to publish data to data streams.

The adapter ignores the **start** command if it is executed when there is a running instance of the adapter, and a warning is sent.

See also

- *Starting the NYSE Adapter* on page 237

Stop Command

The **stop** command disconnects the SpPublisher and SpSubscriber from Event Stream Processor, causes the Data Source Handler to close the session and disconnect from the

datasource, stops the Adapter Controller from listening to user commands, and terminates the adapter process.

If the **stop** command is executed when there is no instance of a running adapter, the command is ignored and a warning is sent.

See also

- *Stopping the NYSE Adapter* on page 239

Status Command

The **status** command reports the adapter status, and the Adapter Controller prints out its status: either running or stopped.

See also

- *Checking the NYSE Adapter Status* on page 238

Watchlists

Dynamically control message flow through the adapter using two watchlist streams: market data and order book.

The adapter subscribes, over different transports, to various symbols from available namespaces. The symbol+ namespace+transport triads are called subscription keys. The watchlists map subscription keys to data streams. The watchlist stream names are defined in the adapter configuration file.

The adapter supports group subscriptions for market data. Group subscriptions are identified by a group symbol, and each group symbol is associated with various individual symbols. Data coming on symbols from the same group are stored in the same data stream. Data stream records are keyed by individual symbols. For example, if symbols X1, X2, and X3 are associated with GROUPX, the data records are keyed using X1, X2, and X3 as opposed to GROUPX.

Subscription keys relate to streams as many to one. Subsequently, a subscription key may not target more than one market data stream and one order book stream per side (Buy or Sell). However, a data stream may be targeted by multiple subscription keys. Order book sides (Buy and Sell) may be hosted on the same or separate streams.

Watchlist inserts and deletes are user-controlled, while updates are interpreted as error conditions. The adapter reacts to changes in watchlists as follows:

Insert	Activates subscription to symbol from specified namespace over specified transport.
Delete	Deactivates subscription.

Update	Logs an error. Sends an alert to operator (as configured). Continues to send data to old data streams. The valid way to modify the target streams is to shut down and restart the adapter.
--------	--

See also

- *Watchlist Stream Configuration Parameters* on page 232
- *Market Data Watchlists* on page 222
- *Order Book Watchlists* on page 223
- *Watchlist Operation* on page 239
- *Insert* on page 239
- *Delete* on page 240

Market Data Watchlists

Example of market data watchlist structure and content.

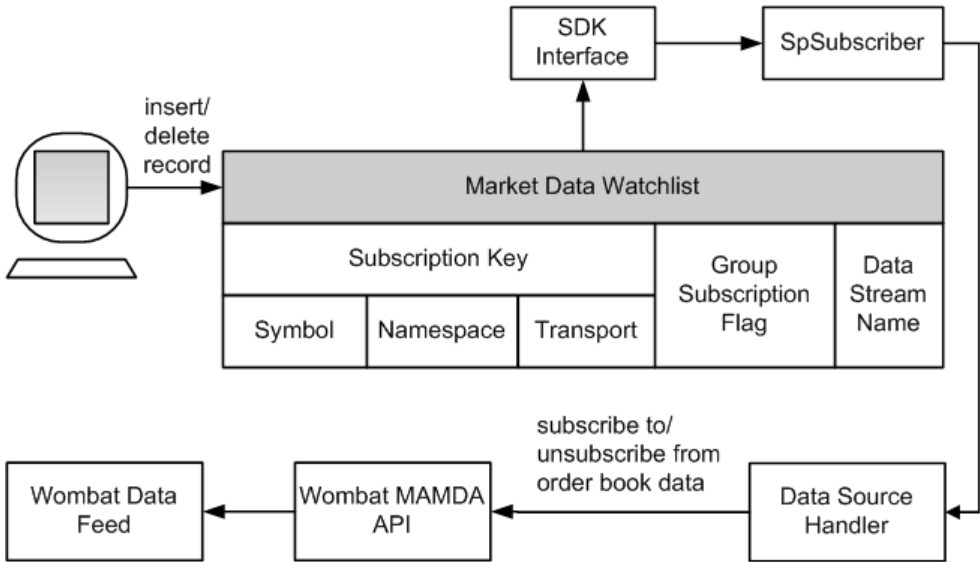


Table 1. Market Data Watchlist Structure and Sample Content

Subscription Key			Group Subscription Flag	Data Stream Name
Sym- bol	Namespace	Transport		
BDK	NASDAQ	T1	false	MarketB
MSFT	NYSE	T1	false	MarketA

Subscription Key			Group Subscription Flag	Data Stream Name
Sym- bol	Namespace	Transport		
IBM	NASDAQ	T2	false	MarketA
GROUP X	NYSE	T2	true	MarketC

See also

- *Watchlist Stream Configuration Parameters* on page 232
- *Order Book Watchlists* on page 223
- *Watchlists* on page 221

Order Book Watchlists

Example of order book watchlist structure and content.

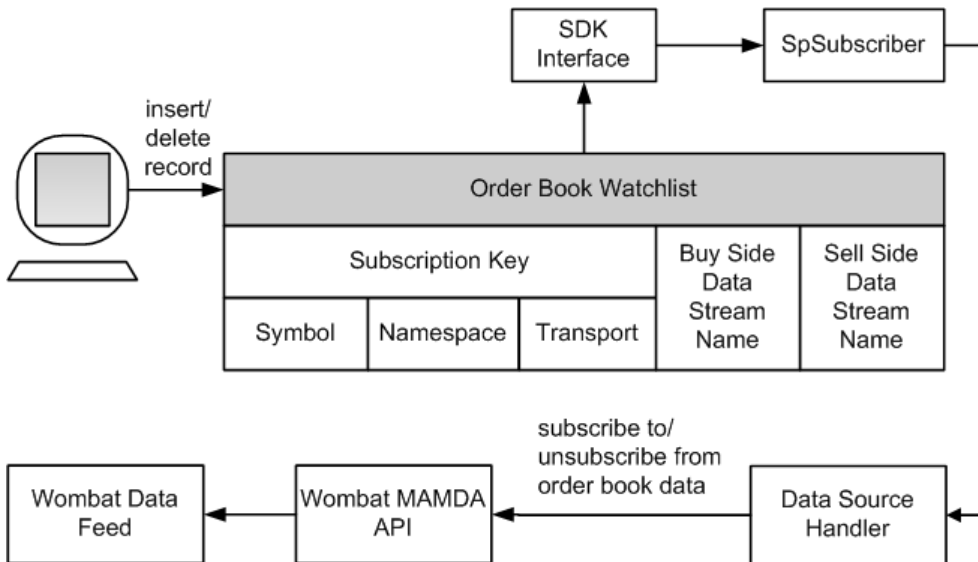


Table 2. Order Book Watchlist Structure and Sample Content

Subscription Key			Buy Side Data Stream	Sell Side Data Stream
Sym- bol	Namespace	Transport		
APPL	NASDAQ	T1	BookABuy	-

Subscription Key			Buy Side Data Stream	Sell Side Data Stream
Sym- bol	Namespace	Transport		
BDK	NASDAQ	T1	BookABuy	BookASell
MSFT	NYSE	T1	BookB	-
IBM	NASDAQ	T2	BookB	BookB

See also

- *Watchlist Stream Configuration Parameters* on page 232
- *Market Data Watchlists* on page 222
- *Watchlists* on page 221

Data Streams

There are two types of data streams: market data and order book.

See also

- *Data Stream Configuration* on page 233

Market Data Streams

A market data stream contains the record key, one or more fields, and the stale flag.

The record key consists of the symbol, namespace (if present), and transport (if present). If you omit the namespace, the transport, or both, incoming symbol updates from different namespaces and transports are stored in the same record.

A market data stream column may have the same name as the hosted field, for example, `wBidSize`, `wBidPrice`. Custom-named columns (for example, `MyTimestamp`) are mapped to field names in the adapter configuration file.

Table 3. Sample Market Data Stream Content

Record Key			wBid- Size	wBid- Price	My Time- stamp	Stale
Sym- bol	Name- space	Trans- port				
MSFT	NYSE	T1	550	33.67	31--12--2008T 10:32:10.536	false
IBM	NASDAQ	T2	430	51.89	31--12--2008T 10:32:44.993	true

Record Key			wBid-Size	wBid-Price	My Time-stamp	Stale
Sym-bol	Name-space	Trans- port				
X1	NYSE	T2	850	133.63	31--12--2008T 10:27:58.563	false
X2	NYSE	T2	440	74.36	31--12--2008T 10:29:03.755	false
X3	NYSE	T2	180	21.53	31--12--2008T 10:31:55.001	false

See also

- *Data Stream Configuration* on page 233

Order Book Data Streams

An order book data stream contains the record key, entry ID, price, total size, timestamp, and stale flag.

The number of price levels is unlimited.

The record key consists of the symbol, namespace (if present), transport (if present), side indicator (if present), and price. If you omit the namespace, transport, or both, incoming symbol updates from different namespaces and transports are consolidated.

Valid values for the side indicator are B (Bid) and A (Ask). The side indicator column is mandatory if the data stream is targeted for both sides of the order book. The adapter ignores the value of the side indicator column for subscriptions that target separate buy and sell streams.

Table 4. Sample Order Book Stream Content

Record Key				EntryID	Size	Timestamp	Stale
Sym-bol	Name-space	Side	Price				
IBM	NASDAQ	B	106.23	25345	200	31--12--2008 T10:32:10.5 36	false
IBM	NASDAQ	B	106.22	74558	300	31--12--2008 T10:32:09.2 11	false
IBM	NASDAQ	B	106.19	12347	600	31--12--2008 T10:32:07.8 40	false

Record Key				EntryID	Size	Timestamp	Stale
Sym- bol	Name- space	Side	Price				
IBM	NASDAQ	A	108.73	53298	200	31--12--2008 T10:32:05.2 66	false
IBM	NASDAQ	A	108.11	53749	300	31--12--2008 T10:32:03.7 54	false
MSFT	NYSE	-	55.93	65228	400	31--12--2008 T10:31:53.9 22	false
MSFT	NYSE	-	55.87	54349	700	31--12--2008 T10:31:46.7 25	false

Note: Transport is ignored when storing data records. Also, the side indicator is empty when its value is derived from the watchlist.

See also

- *Data Stream Configuration* on page 233

Stale Records

Data stream records are marked stale when certain adapter functions fail.

When the adapter starts, it sends finalization instructions to Event Stream Processor. These instructions are run if communication with the adapter is lost or the adapter fails to deliver two consecutive heartbeats. The finalization procedure sets the stale flag to 1 (true) in all records of all configured adapter data streams.

Data stream records are also marked stale if:

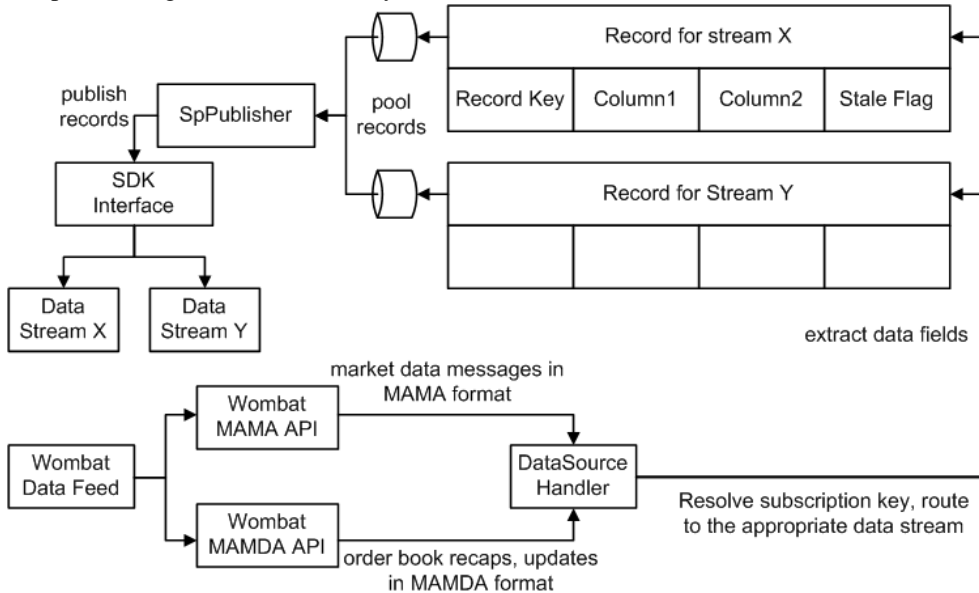
- A subscription is deactivated.
- The adapter receives a market data gap message.
- The adapter receives an order book gap message.
- The adapter receives a transport disconnect message from the NYSE data feed.
- The adapter is about to shut down due to a user command or fatal error condition.

See also

- *Event Stream Processor Parameters* on page 230

Message Flow

Adapter message flow is initiated by the start command.



The Data Source Handler receives real-time market data messages in MAMA format. It receives order book recaps and updates, such as deltas and complex deltas, in MAMDA format. It resolves the subscription key and routes the message to the appropriate data stream, where the data fields extract and convert into stream records.

A record is then ready to be published to Event Stream Processor but is not published immediately. Records are queued, then picked up by SpPublisher. Set the queue capacity in the adapter configuration file. A larger queue is less likely to overflow if a message burst occurs. When the queue is three-quarters full, a warning is logged. Another warning is logged when the queue returns to three-quarters empty. If the queue is full, the adapter waits until room becomes available before it places the next record.

The adapter uses record pooling for performance considerations. The dequeued records are pooled according to user preferences in the adapter configuration file. Messages are published when the pool size reaches the Maximum Pool Size or after a Maximum Pooling Time since the previous publication, whichever occurs first. The Maximum Pooling Time drives the adapter's latency. If the Maximum Pooling Time is too short or Maximum Pool Size is too small, messages are published to Event Stream Processor in undersized batches, resulting in poor overall performance.

When a pooled record batch is ready for publication, the SpPublisher uses the Pub/Sub API to send the records to Event Stream Processor. Records are published asynchronously. The

CHAPTER 2: Adapters Supported by Event Stream Processor

adapter receives no feedback from Event Stream Processor. In the event of a failover, the Pub/Sub API switches to the spare Event Stream Processor instance without message loss.

Datatype Mapping for the NYSE Adapter

Event Stream Processor datatypes map to NYSE datatypes.

Event Stream Processor Data-type	MAMA API Datatype
integer	long
long	long
float, money, money1-money15	double
string	string
bigdatetime	com.wombat.mama.MamaDateTime
interval	com.wombat.mama.MamaDateTime
binary	string

Setting the JAVA_HOME Environment Variable

Set the JAVA_HOME environment variable to point to the Java directory.

Prerequisites

Install Java Runtime Environment version 1.6.0_26 or higher. Place NYSE Java and binary libraries under \$ESP_HOME/adapters/wombat/lib/wombat.

Task

Set the JAVA_HOME environment variable to the directory path where Java Runtime Environment 1.6.0_26 or higher is installed.

Next

Verify that the ESP_HOME environment variable is set correctly.

Configuration

Configuration information for the NYSE Technologies adapter.

NYSE Adapter Directory

The adapter directory contains all files, such as configuration files, templates, examples, and JAR files, relating to the adapter.

README.txt Quick Guide
ReleaseNotes.txt Release Notes

```

bin/
  adapter.bat  Standalone adapter startup script
  adapter.sh   Standalone adapter startup script
  adapter-plugin.bat Plug-in connector startup script
  adapter-plugin.sh Plug-in connector startup script

config/
  controller.xsd      Controller schema
  log4j.properties   Sample logging configuration
  wombatadapter.xsd  Adapter schema
  dictionary.txt      The configuration file to map wombat fields
with ESP datatypes. This is used for data dictionary.
  login.config       Authentication configuration

discovery/          Data discovery templates

example/            Working example

lib/
  esp_wombat_adapter.jar wombat adapter library
  wombat/              wombat java and binary libraries

javadoc/
  adapterapi/         Adapter API Javadoc
  wombatadapter/      Wombat Adapter Javadoc

```

Common jars are located:

```

$ESP_HOME/adapters/jar

activation.jar          Java mail library
adapterapi.jar         Adapter API code
axis.jar               Webservices jar
commons-codec-1.3.jar  Required by SDK API
commons-discovery-0.2.jar
commons-logging-1.1.jar Logging library
esp_java_sdk-0.2.jar   ESP SDK library
jaxrpc-api-1.1.jar     Required by ESP SDK
log4j-1.2.14.jar       Logging library
mail.jar               Java mail library
saaj-api-1.3.jar       Webservices jar
ws-commons-util-1.0.2.jar Required by ESP SDK
wsdl4j-1.5.1.jar
xercesImpl.jar         XML parser library
xmlrpc-client-3.1.3.jar Required by ESP SDK
xmlrpc-common-3.1.3.jar Required by ESP SDK
xmlrpc-server-3.1.3.jar Required by ESP SDK

```

Schema and Configuration File

The adapter configuration is loaded from a file and validated against the adapter schema.

The adapter working example includes a sample `adapter.xml` file. You can edit this file or write a new one. Ensure that the adapter configuration validates against the schema. An error message displays if the configuration does not validate.

Adapter Controller Parameter

The **controllerPort** parameter specifies the adapter command and control port.

Parameter Name	Type	Description
controllerPort	positive integer	(Required) Specifies the adapter command and control port. User commands are sent to this port on localhost.

Event Stream Processor Parameters

Event Stream Processor parameters configure communication between Event Stream Processor and the NYSE adapter.

These parameters are defined in the `controller.xsd` file in the `config` directory.

Parameter Name	Type	Description
espAuthType	string	<p>(Required) Specifies method used to authenticate to the Event Stream Processor. Valid values are:</p> <ul style="list-style-type: none"> server_rsa – RSA authentication using keystore user_password – Kerberos and LDAP authentication none – No authentication <p>If the adapter is operated as a Studio plug-in, espAuthType is overridden by the Authentication Mode Studio start-up parameter.</p>
espUser	string	(Required) Specifies user name required to log in to Event Stream Processor. It is required for any authentication scheme other than none (see espAuthType). No default value.

Parameter Name	Type	Description
espPassword	string	(Required) Specifies the password required to log in to Event Stream Processor. Required for any authentication scheme other than none (see espAuthType). Includes an "encrypted" attribute indicating whether the espPassword value is encrypted. Default value is false. If set to true, the password value is decrypted using espRSAKeyStore and espRSAKeyStorePassword .
espProjectUri	string	(Required) Specifies the total project Uri to connect to Event Stream Processor cluster. For example, <code>esp://localhost:19011/ws1/p1</code> .
pulseInterval	non-negative integer	(Optional) Specifies time interval, in seconds, during which outbound record changes are coalesced by Event Stream Processor, then received by the adapter as a single event. If not set or set to 0, record changes are received individually as they occur.
espHeartbeatPeriod	positive integer	(Optional) Specifies number, in seconds, that adapter waits before sending the next heartbeat to Event Stream Processor. If Event Stream Processor fails to receive two consecutive heartbeats, all records the adapter publishes are marked stale. Default value is 10.
recordQueueCapacity	positive integer	(Optional) Specifies capacity of the record queues. Default value is 4096.
maxPubPoolSize	positive integer	(Optional) Specifies the maximum size of the record pool. Record pooling allows for faster publication. Default value is 256.
maxPubPoolTime	positive integer	(Optional) Specifies 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.

Parameter Name	Type	Description
useTransactions	boolean	(Optional) If set to true, pooled messages are published to Event Stream Processor in transactions. If set to false, they are published in envelopes. Default value is false.
espRSAKeyStore	string	(Dependent required) Specifies the location of the RSA keystore, and is used to decrypt the password value. Required if espAuthType is set to <code>server_rsa</code> , or the encrypted attribute for espPassword is set to true, or both.
espRSAKeyStorePassword	string	(Dependent required) Specifies the keystore password, and is used to decrypt the password value. Required if espAuthType is set to <code>server_rsa</code> , or the encrypted attribute for espPassword is set to true, or both.
espEncryptionAlgorithm	string	(Optional) Used when the encrypted attribute for espPassword is set to true. If left blank, RSA is used as default.

See also

- *Stale Records* on page 226

Watchlist Stream Configuration Parameters

Watchlist stream configuration parameters specify the names of the market data and order book watchlists.

Parameter Name	Type	Description
marketDataWatchlist	string	(Required) Specifies name of the market data watchlist.
orderBookWatchlist	string	(Required) Specifies name of the order book watchlist.

See also

- *Market Data Watchlists* on page 222
- *Order Book Watchlists* on page 223
- *Watchlists* on page 221

Data Stream Configuration

Use the `marketDataStreams` section in the configuration file to provide data stream parameters.

Indicate the stream name for each data stream.

Data stream columns and the corresponding MAMA fields may have the same or different names. If the names are different, map the column and its corresponding data field explicitly. In the example below, the `MyTimestamp` column is mapped to the `wSrcTime` MAMA field.

```
<column>
<name>MyTimestamp</name>
<field>wSrcTime</field>
</column>
```

Ensure columns have the same data type as their corresponding fields. Some columns may correspond to no field. Column names `Symbol`, `Namespace`, `Transport`, and `Stale` are reserved.

See also

- *Data Streams* on page 224
- *Market Data Streams* on page 224
- *Order Book Data Streams* on page 225

Datafeed Parameters

Datafeed parameters configure the datafeed for the NYSE adapter.

Refer to the *MAMA Developer's Guide* for more detailed information about these parameters.

Parameter Name	Type	Description
middleware	string	(Required) Specifies the name of the middleware API. Currently only Configuration 9 <code>wmw</code> (NYSE TCP Middleware) is supported. Default value is <code>wmv</code> or <code>lbn</code> .
subscriptionTimeout	positive integer	(Required) Specifies number of seconds the adapter waits, without receiving an initial value, before it resends a market data subscription request.
subscriptionRetries	positive integer	(Required) Specifies how many attempts to make to obtain an initial image for a subscription.
dictionaryTransport	string	(Required) Specifies transport over which the MAMA dictionary is requested on adapter startup.

Parameter Name	Type	Description
dictionaryNamespace	string	(Required) Specifies namespace from which the MAMA dictionary is requested on adapter startup.
dictionaryTimeout	positive integer	(Required) Specifies number of seconds the adapter waits, without receiving a response, before it resends a MAMA dictionary request.
dictionaryRetries	positive integer	(Required) Specifies how many attempts to make to obtain the MAMA dictionary.

Sample NYSE Configuration File

Sample configuration file (`adapter.xml`) for the NYSE adapter.

This file is in the `example` folder.

```
<adapter>
<!-- Adapter Controller -->
<controller>
  <controllerPort>13579</controllerPort>
</controller>

<!-- Event Stream Processor Settings -->
<esp>
  <espConnection>
    <espProjectUri>esp://localhost:19011/w1/pl</espProjectUri>
  </espConnection>

  <espSecurity>
    <espUser>espuser</espUser>
    <espPassword encrypted="false">espuser</espPassword>
    <espAuthType>none</espAuthType>
  <!--
    <espRSAKeyStore>/keystore/keystore.jks</espRSAKeyStore>
    <espRSAKeyStorePassword>Sybase123</espRSAKeyStorePassword> --
  >
    <espEncryptionAlgorithm>RSA</espEncryptionAlgorithm>
  </espSecurity>
  <maxPubPoolSize>1</maxPubPoolSize>
</esp>

<watchlists>
  <marketDataWatchlist>MarketDataWatchlist</marketDataWatchlist>
  <orderBookWatchlist>OrderBookWatchlist</orderBookWatchlist>
</watchlists>

<marketDataStreams>
  <stream>
    <name>MyMarketDataStream</name>
    <column>
```

```

    <name>MyTimestamp</name>
    <field>wSrcTime</field>
  </column>
</stream>
</marketDataStreams>

<datafeed>
  <middleware>wmw</middleware>
  <subscriptionTimeout>5</subscriptionTimeout>
  <subscriptionRetries>1</subscriptionRetries>
  <dictionaryTransport>demo</dictionaryTransport>
  <dictionaryNamespace>WOMBAT</dictionaryNamespace>
  <dictionaryTimeout>10</dictionaryTimeout>
  <dictionaryRetries>1</dictionaryRetries>
</datafeed>

</adapter>

```

NYSE Input Adapter

The NYSE Input adapter connects to a NYSE data feed to receive real-time level 1 and level 2 market data.

You can configure the adapter on any source stream as an input data location. The authentication method is set to that of Event Stream Processor: none, rsa, or gssapi. This adapter supports schema discovery.

To use this adapter, ensure NYSE adapter version 1 or later is installed.

Property Label	Property ID	Type	Description
Connector Directory Path	baseDir	directory	(Required) Specify the path to the adapter installation directory. This property is ignored if the Connector Remote Directory Path property is supplied. Default value is <code>/mydir/NYSEA-adapter</code> .
Configuration File Path	configFilePath	configFilename	(Required) Specify the absolute path to the adapter configuration file. This property is ignored if the Remote Configuration File Path property is supplied. Default value is <code>/mydir/NYSEA-adapter/config/adapter.xml</code> .

Property Label	Property ID	Type	Description
Discovery Directory Path	discDirPath	directory	(Required) Specify the absolute path to the adapter discovery directory. Default value is /my-dir/NYSEadapter/discovery.
Connector Remote Directory Path	remoteBaseDir	string	(Advanced) Specify the path to the adapter remote base directory (for remote execution only). If this property is supplied, the Connector Directory Path property is ignored.
Remote Configuration File Path	remoteConfigFilePath	string	(Advanced) Specify the absolute path to the adapter configuration file (for remote execution only). If this property is supplied, the Configuration File Path property is ignored.
Discovered Table (runtime)	table	tables	(Advanced) Name of the discovered table. This is filled in by Studio.
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.

Logging

The NYSE adapter uses the Apache `log4j` API to log errors, warnings, and information and debugging messages.

The `log4j.properties` file contains the logging configuration. A sample of this file is included in the adapter distribution.

Note: Setting the logging level to DEBUG may result in very large log files. The default level is INFO. Raw Wombat messages are logged at the DEBUG level.

Refer to <http://logging.apache.org/log4j> for details on logging configuration.

Operation

Operate the NYSE adapter from the command line.

Ensure the project to be run contains the market data and order book watchlists. Check that the names of the watchlist streams correspond to the **marketDataWatchlist** and **orderBookWatchlist** parameters respectively.

Set the desired logging level in `log4j.properties`.

Starting the NYSE Adapter

To start the NYSE adapter from the command line, start Event Stream Processor and execute the **start** command.

1. Start Event Stream Processor.

Windows:

1. Start the example cluster.

```
cd %ESP_HOME%\cluster\nodes\node1
%ESP_HOME%\bin\esp_server --cluster-node node1.xml
```

2. Compile CCL to create CCX.

```
%ESP_HOME%\bin\esp_compiler -i model.ccl -o model.ccx
```

3. Deploy the project on the cluster.

```
%ESP_HOME%\bin\esp_cluster_admin" --uri=esp://localhost:19011
--username=sybase --password=sybase --add_project --workspace-
name=w1 --project-name=p1 --ccx=model.ccx
```

4. Start the deployed project on the cluster.

```
%ESP_HOME%\bin\esp_cluster_admin" --uri=esp://localhost:19011
--username=sybase --password=sybase --start_project --
workspace-name=w1 --project-name=p1
```

UNIX:

1. Start the example cluster.

```
cd $ESP_HOME/cluster/nodes/node1
$ESP_HOME/bin/esp_server --cluster-node node1.xml
```

2. Compile CCL to create CCX.

```
$ESP_HOME/bin/esp_compiler -i model.ccl -o model.ccx
```

3. Deploy the project on the cluster.

```
$ESP_HOME/bin/esp_cluster_admin" --uri=esp://localhost:19011
--username=sybase --password=sybase --add_project --workspace-
name=w1 --project-name=p1 --ccx=model.ccx
```

4. Start the deployed project on the cluster.

CHAPTER 2: Adapters Supported by Event Stream Processor

```
$ESP_HOME/bin/esp_cluster_admin" --uri=esp://localhost:19011  
--username=sybase --password=sybase --start_project --  
workspace-name=w1 --project-name=p1
```

2. Start the adapter.

Operating System	Step
UNIX	Open a terminal window and enter: cd \$ESP_HOME/adapters/wombat/bin ./adapter.sh <configuration file path> start
Windows	Open a command window and enter: cd %ESP_HOME%/adapters/wombat/bin adapter.bat <configuration file path> start

You can use the **esp_subscribe** utility to ensure that NYSE messages are successfully published to Event Stream Processor.

See also

- *Start Command* on page 220

Checking the NYSE Adapter Status

To check the NYSE adapter status from the command line, execute the **status** command.

Operating System	Step
UNIX	Open a terminal window and enter: cd \$ESP_HOME/adapters/wombat/bin ./adapter.sh <configuration file path> status
Windows	Open a command window and enter: cd %ESP_HOME%/adapters/wombat/bin adapter.bat <configuration file path> status

You see the adapter status, which is either running or stopped.

See also

- *Status Command* on page 221

Stopping the NYSE Adapter

To stop the NYSE adapter from the command line, execute the **stop** command.

Operating System	Step
UNIX	Open a terminal window and enter: <pre>cd \$ESP_HOME/adapters/wombat/bin ./adapter.sh <configuration file path> stop</pre>
Windows	Open a command window and enter: <pre>cd %ESP_HOME%/adapters/wombat/bin adapter.bat <configuration file path> stop</pre>

See also

- *Stop Command* on page 220

Watchlist Operation

Watchlists can be modified using inserts and deletes.

Watchlist updates are interpreted as error conditions and no action is taken.

Modifying the market data watchlist causes the adapter to subscribe to or unsubscribe from real-time data on symbols. Modifying the order book watchlist causes the adapter to subscribe to or unsubscribe from order book data on symbols.

See also

- *Watchlists* on page 221

Insert

A watchlist insert triggers two actions in the adapter: subscribing and publishing.

A watchlist insert triggers these actions:

- The adapter subscribes to the specified symbol from the specified namespace over the specified transport.
- The adapter receives real-time market data messages or order book recaps and updates, and publishes them to the corresponding data streams.

See also

- *Watchlists* on page 221

Delete

A watchlist delete triggers two actions in the adapter: unsubscribing and marking records stale.

A watchlist delete triggers these actions:

- The adapter unsubscribes the specified symbol from the specified namespace over the specified transport.
- Market data stream records that result from the canceled subscription are marked stale.

See also

- *Watchlists* on page 221

Example: Subscribing to and Publishing Data

Subscribe to real-time market data on two symbols and order book data on one symbol, and publish the incoming data to Event Stream Processor.

Prerequisites

You have a network connection to the NYSE datafeed.

Task

1. Edit the `adapter.sh` script.
2. Set the `JAVA_HOME` environment variable to the directory where Java Runtime Environment (JRE) is installed.

Note: The NYSE libraries are available in both 32- and 64-bit versions. If your libraries are 32-bit, use a 32-bit JRE. If your libraries are 64-bit, use a 64-bit JRE.

3. Start Event Stream Processor.

Operating System	Step
UNIX	Open a terminal window: <ol style="list-style-type: none"> 1. Start the example cluster: <code>start_server_cluster.sh</code> 2. Start the project on the cluster: <code>start_project.sh</code>
Windows	Open a command window: <ol style="list-style-type: none"> 1. Start the example cluster: <code>start_server_cluster.bat</code> 2. Add project to the cluster, and start it on the cluster: <code>start_project.bat</code>

4. Edit the `mama.properties` file in the adapter `lib/wombat` directory to ensure the `subscribe_address` and `subscribe_port` properties point to a NYSE data feed.

5. Start the adapter.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./adapter.sh</code>
Windows	Open a command window and enter: <code>adapter.bat</code>

6. Wait five to ten seconds for the adapter to initialize.
7. Upload a stream record.

Operating System	Step
UNIX	<code>./upload.sh</code>
Windows	<code>upload.bat</code>

8. Start the subscriber utility to view data stream content.

Operating System	Step
UNIX	<code>./esp-subscribe.sh</code>
Windows	<code>esp-subscribe.bat</code>

Open Adapter

The Sybase Event Stream Processor Open adapter is a version of the open-source `openadapter`[™] (openadapter.org).

A range of adapters is available for common applications and middleware environments, such as Web services and various file types. Each adapter can be used with a variety of readers and writers to parse and format different types of messages (for example, delimited field records or XML documents). The records coming in through the adapter can include an `ESP_OPS` column that indicates the database operation to perform with the record.

- `i, I, insert, or INSERT` indicates an insert.
- `p, P, upsert, or UPSERT` indicates an upsert.
- `u, U, update, or UPDATE` indicates an update.
- `s, S, safedelelete, or SAFEDELELETE` indicates a safedelelete.
- `d, D, delete, or DELETE` indicates a delete.

If you use the `ESP_OPS` column, ensure every record in this column has a value.

An Open adapter is defined by an adapter properties file, and includes a number of components that move data from one or more source components to one or more sink components. You may also configure intermediate components (pipes) to perform additional

CHAPTER 2: Adapters Supported by Event Stream Processor

processing on the data. In a system where a number of possible adapters can run, each adapter runs as a separate instance that you start and control individually.

Note: On Microsoft Windows, use double backslash \\ as separator in paths, class paths, and URLs.

The Open adapter requires a separately purchased license. This license supports the standard SySAM grace period, meaning it can run unlicensed for 30 days. After this period, the adapter cannot run without a valid license.

If you purchased your product from Sybase or an authorized Sybase reseller, go to the secure Sybase Product Download Center (SPDC) at <https://sybase.subscribenet.com> and log in to generate license keys. The license generation process may vary slightly, depending on whether you ordered directly from Sybase or from a Sybase reseller.

If you ordered your product under an SAP contract and were directed to download from SAP Service Marketplace (SMP), you can use SMP at <http://service.sap.com/licensekeys> to generate license keys for Sybase products that use SySAM 2-based licenses.

Datatype Mapping for the Open Adapter

Event Stream Processor datatypes map to Open adapter datatypes.

Event Stream Processor Datatype	Open Adapter Datatype
integer	integer
long	long
float	double
date	datetime
timestamp	datetime
bigdatetime	long
boolean	short
interval	long
binary	string
money	double
money(1-n)	double
string	string

Setting the JAVA_HOME Environment Variable

Set the JAVA_HOME environment variable to point to the Java directory.

Set the JAVA_HOME environment variable to the directory path where Java Runtime Environment 1.6.0_26 or higher is installed.

Next

Verify that the ESP_HOME environment variable is set correctly.

Configuration

The adapter properties (.props) files are text files that contain configuration information for the components to be invoked for an adapter.

You can create properties files by using a text editor or the Adapter Framework Editor. A configuration can contain any number of source, sink, and pipe components, and their respective readers and formatters. The Sybase Open adapter can also read properties from XML documents.

Each adapter properties file contains the configuration for a single adapter. An adapter property specifies the adapter, component, and property names:

```
AdapterName.ComponentName.PropertyName=PropertyValue
```

For example:

A # character in the first column denotes a comment line.

```
#
# Adaptor 'Dynamic2' Component 'BalanceInMQ'
#
Dynamic2.BalanceInMQ.QueueManager=QM_Test
Dynamic2.BalanceInMQ.Queue=TEST.BALANCE.IN ... #
# Adaptor 'Dynamic2' Component 'EventInMQ'
#
Dynamic2.EventInMQ.QueueManager=QM_Test
Dynamic2.EventInMQ.Queue=TEST.EVENT.IN
```

Properties may be qualified to more levels. Ensure that properties that define a field are defined for both the field name and the field type. For example:

```
Dynamic2.EventInMQ.Field1=Date
Dynamic2.EventInMQ.Date.Name=CurrentDate Dynamic2.
EventInMQ.Date.Type=Datetime
```

The Open adapter supports properties for individual components. You can use a number of statements in the adapter properties file to simplify the definition of properties.

Open Adapter Directory

The adapter directory contains all files, such as configuration files, templates, examples, and JAR files, relating to the adapter.

```
$ESP_HOME/adapters/esp_open  Root directory for the Open adapter.
This directory contains the log4j.xml configuration file

  lib/ All adapter and third-party distributable jar files.
  bin/ Example scrips for starting the adapters and editor.
  repo/ Standard location for all property files.
  examples/ Various component examples.
```

Include Files Syntax

Syntax for including an additional properties file.

```
#include other.props
```

The file name can be preceded by a prefix, which is added to each property name in the included file:

```
#include A.comp other.props
```

Where `other.props` contains:

```
property1=foo
```

The Open adapter reads:

```
A.comp.property1=foo
```

Variable Substitution

Defines a variable within the properties file or an included properties file.

You can define a variable within the properties file or an included properties file if the variable is defined before it is used:

```
NUM_TO_SEND=1000 ... A.comp.MaxRecords=${NUM_TO_SEND}
```

You can also define the variable using the "-D" option to the Java Virtual Machine when the adapter is started:

```
java -DNUM_TO_SEND=1000 org.openadapter.adapter.RunAdapter
config.props A
```

Wildcard Property Names

If a component initializes and attempts to get property values for a property that is defined with a wildcard name, the `SuperProperties` class returns the value for the wildcard property unless there is a more specific property setting that matches the request.

For example, this matches any adapter and component names:

```
*.QueueManager=QM_Test
```

This matches exactly one component in the name:

```
A.?.QueueManager=QM_Test
```

A.B.QueueManager matches; however, A.B.C.QueueManager does not.

Autoincremented Property Names

You can choose to autoincrement property names if there is a long list of properties to be specified.

For example:

```
A.comp.field1=foo A.comp.field2=bar A.comp.field3=hello
A.comp.field4=world
```

You can autoincrement these fields:

```
A.comp.field++=foo A.comp.field++=bar A.comp.field++=hello
A.comp.field++=world
```

Note: If you use the Adaptor Framework Editor, the autoincrement fields convert to the corresponding numbered fields on reading the configuration file. The Adaptor Framework Editor cannot revert the fields to auto-numbered fields on rewriting the properties file.

XML Properties Files

Specify adapter properties files as XML documents.

```
<?xml version="1.0" encoding="UTF-8" ?>
<openadapter>
  <A>
    <Component1>
      <Name>C1</Name>
    </Component1>
    <Component2>
      <Name>C2</Name>
    </Component2>
    <C1>
      <ClassName>com.sybase.adapter.ibm.MqSource</ClassName>
    </C1>
    <C2>
      ...
    </C2>
  </A>
</openadapter>
```

Open Adapter Components

Add at least one source and one sink component to use the Open adapter. Source components read provided data, and sink components write to associated output.

Each component has its own required properties. Set the `DOSTringReader` and `DOSTringWriter` properties for the source and sink components to enable data passing through these components to be parsed and formatted by various parsers and formatters.

Event Stream Processor does not support multibyte character sets, such as UTF-16. However, an external source can contain non-ASCII characters. By default, the adapter interprets them as 1-byte or 2-byte Unicode characters, which may lead to data corruption. To set the encoding explicitly, add the `TextEncoding` property to the configuration file. For example:

```
Adapter.Component.TextEncoding=ASCII
```

If a property for defining multiple table names is specified as:

```
Adapter.Component.Table++=TableName
```

the configuration file contains:

```
Adapter.Component.Table1=TableName1  
Adapter.Component.Table2=TableName2
```

You can define properties with a number of levels separated by a period. For example, a property specific to `Table1` can be represented as:

```
Adapter.Component.TableName1.Field1=FieldName1
```

Source Components

The Open adapter has two source components: `AsapSource` and `SpPersistentSubscribeSource`.

AsapSource Properties

The `AsapSource` component subscribes to data from the Event Stream Processor stream name specified in the adapter configuration.

```
ClassName=com.sybase.esp.adapter.asap.AsapSource
```

Property	Description
<code>ProjectUri</code>	(Dependent required) Connect to the Server running in cluster mode. For example, <code>esp://localhost:19011/ws1/p1</code> .
<code>User</code>	(Required) The initial connection between <code>AsapSource</code> and Event Stream Processor requires authentication. Enter a valid user name known to Event Stream Processor.

Property	Description
Passwd	<p>(Required) The initial connection between AsapSource and Event Stream Processor requires authentication. Enter a valid password for the user name configured above.</p> <hr/> <p>Note: If the <code>IsEncrypted</code> property is set to true, the user and password information is passed to Event Stream Processor before the SSL connection is set up. These details are passed in plain text.</p>
IsEncrypted	(Optional) If this property is set to true, AsapSource attempts to use an SSL socket connection to Event Stream Processor.
UseServerRSA	<p>(Optional) If true, server RSA authentication is used to connect to the Server. If you specify this property, also provide the <code>KeyStore</code> and <code>KeyStorePassword</code> properties.</p> <p>The Open adapter uses the Bouncy Castle Java security implementation. Ensure that Bouncy Castle is listed among other security providers in the <code>java.security</code> file of your Java Runtime Environment directory:</p> <pre>=org.bouncycastle.jce.provider.BouncyCastleProvider</pre>
KeyStore	(Optional) Used for server RSA authentication. Specifies the location of the keystore (.jks file). Set this property if you specify <code>UseServerRSA</code> .
KeyStorePassword	(Optional) Specify the keystore password. This is used for server RSA authentication. Set this property if you specify <code>UseServerRSA</code> .
UseUserPassword	(Optional) If set to true, <code>User</code> and <code>Password</code> authentication (for example, Kerberos) is used to connect to the Server. If you specify this, provide values for <code>User</code> and <code>Password</code> .
Stream++	(Optional) Name of stream to subscribe to.
MaxBlockBuildTime (milliseconds)	(Optional) AsapSource sends records in blocks for better performance. A block is built of individual records until the <code>MaxBlockSize</code> is reached or <code>MaxBlockBuildTime</code> elapses, whichever occurs first. Then, the block is sent along the adapter pipeline. If set to 0, there is no limit on the block building time. Default value is 1000 (milliseconds).
MaxBlockSize	(Optional) AsapSource sends records in blocks for better performance. A block is built of individual records until the <code>MaxBlockSize</code> is reached or <code>MaxBlockBuildTime</code> elapses, whichever occurs first. Then the block is sent along the adapter pipeline. Default value is 256.

Property	Description
RecordBufferCapacity	<p>(Optional) If set to 0, records are sent along the adapter pipeline one at a time.</p> <p>If set to a positive number, AsapSource queues the records made available by the Event Stream Processor in an internal buffer.</p> <p>Buffered records are added to ongoing blocks on a dedicated thread so that the Pub/Sub subscription thread can continue buffering the records. When the buffer capacity is exceeded, the queue blocks until the buffer capacity becomes available again. Default value is 4096.</p>
PulseInterval	<p>(Optional) The number of seconds to wait until AsapSource gets the next record from Event Stream Processor. All updates to a record that are made on the Event Stream Processor during the pulse interval are coalesced and only the resulting record is sent.</p> <p>By default, records are received by the adapter instantaneously, rather than being pulsed.</p>

See also

- *Example: Using the AsapSource Component* on page 291

SpPersistentSubscribeSource Properties

The SpPersistentSubscribeSource component subscribes to a stream in Event Stream Processor and sends records on to other components.

```
ClassName =
com.sybase.esp.adapter.asap.SpPersistentSubscribeSource
```

The stream the component subscribes to does not explicitly remove records until asked by the subscriber. Once records are processed, SpPersistentSubscribeSource publishes tags back to the Event Stream Processor to remove rows from the subscribed stream.

SpPersistentSubscribeSource has two additional streams: log stream and truncate stream. For example, you can have three streams named Stream1, Stream1_log, and Stream1Truncate. The log stream has two additional columns: sequence number and opcode. Records pass from Stream1 to Stream1_log, as well as increasing sequence number values. The opcode values in the opcode column in Stream1_log are "insert". After SpPersistentSubscribeSource subscribes to a batch of data (or a single record), the last sequence number of the records is published to Stream1Truncate, which then removes any records prior to that sequence number from the Stream1_log and persistent store (for example, file on hard disk).

CHAPTER 2: Adapters Supported by Event Stream Processor

Property	Description
Host	(Required) Host name for the Event Stream Processor command and control process.
Port	(Required) Port number for the Event Stream Processor command and control process.
ProjectUri	(Dependent required) Connect to the Server running in cluster mode. For example, <code>esp://localhost:19011/wsl/pl</code> .
IsEncrypted	(Optional) If this property is set to true, <code>AsapSource</code> attempts to use an SSL connection to Event Stream Processor.
UseServerRSA	(Optional) If true, server RSA authentication is used to connect to the Server. If you specify this property, also provide the <code>KeyStore</code> and <code>KeyStorePassword</code> properties.
KeyStore	(Optional) Used for server RSA authentication. Specifies the location of the keystore (.jks file). Set this property if you specify <code>UseServerRSA</code> .
KeyStorePassword	(Optional) Specify the keystore password. This is used for server RSA authentication. Set this property if you specify <code>UseServerRSA</code> .
UseUserPassword	(Optional) If set to true, <code>User</code> and <code>Password</code> authentication (for example, Kerberos) is used to connect to the Server. If you specify this, provide values for <code>User</code> and <code>Password</code> .
SyncBaseStreams	(Optional) Asks platform for <code>SYNC_BASE_STREAMS</code> . When set to true, the <code>publisher.commit()</code> method is called after each batch is published to the platform. Default value is true.
CommitLimit	(Optional) Max size of batch to process in one call. Default value is 100.
User	(Required) The initial connection between <code>SpPersistentSubscribeSource</code> and the Event Stream Processor requires authentication. Enter a valid username known to the Event Stream Processor.
Passwd	(Required) The initial connection between <code>SpPersistentSubscribeSource</code> and the Event Stream Processor requires authentication. Enter a valid password for the username configured above.
Stream++	(Optional) Name of stream to subscribe to.

Property	Description
<code><Stream+>.TruncateStream</code>	(Required) Stream responsible for truncating data.
<code><Stream++>.OpcodeColumn</code>	(Required) Name for the column where the opcode value is stored.
<code><Stream++>.SequenceColumn</code>	(Required) Name for the column where the sequence number is stored.
QueueCapacity	<p>(Optional) <code>SpPersistentSubscribeSource</code> queues the records made available by the Event Stream Processor. The queued records are consumed by a separate thread. This property sets the capacity of the internal queue. When the queue is full, the adapter waits for space to become available. The default value is 4096.</p> <hr/> <p>Note: If the <code>IsEncrypted</code> property is set to true, the user and password information is passed to the Event Stream Processor before the SSL connection is set up. These details are passed in plain text.</p>

See also

- *Example: Using the `SpPersistentSubscribeSource` Component* on page 296

Sink Components

The Open adapter has two sink components: `AsapSink` and `WSSink`.

AsapSink Properties

The `AsapSink` component takes records from the source and delivers them to Event Stream Processor.

Ensure that every input adapter configuration includes exactly one `AsapSink` component.

```
ClassName=com.sybase.esp.adapter.asap.AsapSink
```

Property	Description
<code>ProjectUri</code>	(Dependent required) Connect to the Server running in cluster mode. For example, <code>esp://localhost:19011/ws1/p1</code> .
<code>User</code>	(Required) The initial connection between <code>AsapSink</code> and Event Stream Processor requires authentication. Enter a valid user name known to Event Stream Processor.

CHAPTER 2: Adapters Supported by Event Stream Processor

Property	Description
Passwd	<p>(Required) The initial connection between AsapSink and Event Stream Processor requires authentication. Enter a valid password for the user name configured above.</p> <hr/> <p>Note: If the <code>UseSSL</code> property is set to true, the user and password information is passed to the Event Stream Processor before the SSL connection is set up. These details are passed in plain text.</p>
Stream++	(Required) The name of the stream to which the data is delivered.
IsEncrypted	(Optional) If present and set to true, AsapSink attempts to use an SSL connection to Event Stream Processor.
UseServerRSA	(Optional) If true, server RSA authentication is used to connect to the Server. If you specify this property, also provide the <code>KeyStore</code> and <code>KeyStorePassword</code> properties.
KeyStore	(Optional) Used for server RSA authentication. Specifies the location of the keystore (.jks file). Set this property if you specify <code>UseServerRSA</code> .
KeyStorePassword	(Optional) Specify the keystore password. This is used for server RSA authentication. Set this property if you specify <code>UseServerRSA</code> .
UseUserPassword	(Optional) If set to true, <code>User</code> and <code>Password</code> authentication (for example, Kerberos) is used to connect to the Server. If you specify this, provide values for <code>User</code> and <code>Password</code> .
SHINE	(Optional) Shine through is used for update and upsert operations. If this is set to true and a field value is set to null, then an update to an existing record does not affect the value of that field.
PublishMethod	(Optional) Determines how the data is published to the Event Stream Processor: <code>RECORD</code> , <code>COLLECTION</code> , <code>ENVELOPE</code> , <code>TRANSACTION</code> .
SyncBaseStreams	<p>(Optional) Asks platform for <code>SYNC_BASE_STREAMS</code>. When set to true, the <code>publisher.commit()</code> method is called after each batch is published to the platform.</p> <p>Default value is false.</p>
DiscardFile	(Optional) Name of the file where discarded SDOs should land.
TruncateDiscardFile	(Optional) If set to true, the file is truncated.

Property	Description
EspOpsColumn	(Optional) If set, the value of the ESP_OPS attribute in the incoming records is written to the corresponding column and the record is treated as UPSERT regardless of the ESP_OPS value.

See also

- *Example: Using the AsapSink Component* on page 289

WSSink Properties

The WSSink component is a client implementation of a Web service, allowing communication with remote services.

WSSink is consistent with the WSDL descriptor in `lib/WEB-INF/WSAdapterSource.wsdl`.

Note: Build server site Web services based on the WSDL descriptor located in `lib/WEB-INF/WSAdapterSource.wsdl` of the adapter installation directory.

The Web service client uses simple objects called Data Transfer Objects (DTO) as data containers. The classes used are:

1. `com.sybase.adapter.soap.DataTransferObject`

```
public class DataTransferObject {
    private String name;
    private int opcode;
    private Object[] data;
}
```

Ensure the structure of the data field is the same as that defined in the Web service. You can obtain metadata for DTOs.

2. `com.sybase.adapter.soap.DTOMetaData`

```
public class DTOMetaData {
    private String name;
    private DTOAttribute[] attributes;
}
```

which uses class: `com.sybase.adapter.soap.DTOAttribute`

```
public class DTOAttribute {
    private String name;
    private String xsdType;
}
```

`DataTransferObject`, `DTOMetaData`, and `DTOAttribute` all offer getter and setter methods. You can also obtain object definitions from the WSDL descriptor of the service.

Property	Description
URL	(Required) URL string of the server Web service. Example of valid value is "http://eult121.sybase.com:9085/services/WSAdapter-Source".
TypeN	(Required) Name of the message type with which the source component transmits data. TypeN is also the name of the exposed DTO.
TypeN.<DOType>	(Required) Name of the DTO configured on the remote service.
ManualMapping	(Optional) If true, sink uses mapping AttName->DtoAttName given in the configuration file. If false, sink gets DTO information from the service and assumes that all attribute names defined in DTO are also present in the incoming adapter message.
TypeN.AttName++	(Dependent optional) Name of the field as passed by the source within the adapter. It is also the name of the DTO attribute.
TypeN.<AttName++>. DTOAttName	(Dependent optional) Name of attribute defined by remote Web service for the selected DTO type.
DiscardedLogger-Name	(Optional) Name of logger responsible for logging any records that were not processed successfully.

See also

- *Example: Using the WSSink Component* on page 298

Pipe Components

The Open adapter has two pipe components: BeanShellPipe and JDBCLookupPipe.

BeanShellPipe Properties

A scriptable pipe used to modify a message.

You can script the entire message or each field individually in Java, and use this component within incoming and outgoing flows. The scripting is implemented through BeanShell. Refer to <http://www.beanshell.org> for details on this language.

ClassName: `com.sybase.esp.adapter.scripting.BeanShellPipe`

Property	Description
MsgPreProcessor	<p>(Required) BeanShell script for this message. The script is applied to the message before any fields are processed.</p> <p>The script has access to a message object type. The name of the variable is <i>message</i>. For example:</p> <pre>System.out.println("Message received; SDO array size= " + message.peekDataObjects().length);</pre>
MsgPostProcessor	<p>(Required) BeanShell script for this message. The script is applied to the message after all fields have been processed.</p> <p>The script has access to a message object type. The name of the variable is <i>message</i>. For example:</p> <pre>System.out.println("Message sent; SDO array size= " + message.peekDataObjects().length);</pre>
Type++	<p>(Required) The type of the data object that is received from the source component. For an incoming flow (one flowing into Event Stream Processor through an AsapSink component), this is the Event Stream Processor Base Stream name to be updated with the message.</p> <p>For an outgoing flow (one originating from published data from an Event Stream Processor stream), this is the Event Stream Processor stream name that is publishing the data.</p>
.PreProcessor	<p>(Optional) BeanShell script for this message type. The script is applied to the message before any fields are processed.</p> <p>The script has access to a SimpleDataObject object type. The name of the variable is <i>sdo</i>. <i>TypeN</i> is the name of the message type or stream as defined in the associated <i>Type</i> property. For example:</p> <pre>System.out.println("Got data for message type: " + sdo.getType().getName());</pre>
.PostProcessor	<p>(Optional) BeanShell script for this message type. The script is applied to the message after all fields have been processed.</p> <p>The script has access to a SimpleDataObject object type. The name of the variable is <i>sdo</i>. <i>TypeN</i> is the name of the message type or stream as defined in the associated <i>Type</i> property. For example:</p> <pre>System.out.println("Sending data for message type: " + sdo.getType().getName());</pre>

Property	Description
<code>.AttName++</code>	(Required) Field names contained in the associated message type. Bean-Shell scripting is required for this. If the field name does not exist in the received message type, a new field is created. <code>Type</code> is the name of the message type or stream as defined in the associated <code>Type</code> property.
<code>.Script</code>	(Optional) BeanShell script for this field. The script has access to a <code>SimpleDataObject</code> object type. The name of the variable is <code>sdo</code> . <code>Type</code> is the name of the message type/stream as defined in the associated <code>Type</code> property.
<code>AttNameex</code>	(Optional) Field name as defined in the associated <code>AttName</code> property. For example: <pre>if (sdo.isPresent("Amount ") && sdo.getAttributeValue ("Amount")>0 {value="AC";} else {value="CO" }</pre>

See also

- *Example: Using the BeanShellPipe Component on page 292*

JDBCLookupPipe Properties

The JDBCLookupPipe component queries a database at startup and uses the cached result set as a lookup table.

`ClassName: com.sybase.esp.adapter.jdbc.JDBCLookupPipe`

Each record in the lookup table consists of a unique lookup key and an array of added attributes. The lookup key consists of one or more attributes. When a data object arrives from the source:

- The values of the key attributes are matched against a record in the lookup table.
- If no record matches, the data object is passed on to the sink without any transformation.
- If a record in the lookup table does match the value of the key attributes, the added attributes from the lookup table are added to the record, and the record result is passed on to the sink.

Property	Description
<code>JdbcDriver</code>	(Required) The JDBC driver used to connect to the database. For example: <pre>oracle.jdbc.OracleDriver</pre>

Property	Description
JdbcUrl	(Required) The location of the database. For example: <code>jdbc:oracle:thin:@myhost.com:1521:mydatabase</code>
DBProperty++	(Optional) Name of a database property that the pipe sets when connecting to the database. For example, the user name, password, database name, and so on.
DBPropertyn.Value	(Dependent optional) Value for the associated DBProperty. Set this property if the DBProperty++ property is set.
Table	(Required) Name of the database table where lookup is performed.
KeyAttName++	(Required) Attribute names that make up the lookup key.
KeyDbCol++	(Required) Names of the database columns that correspond to KeyAttNames.
ValueAttName++	(Required) Names of the attributes used for added values.
ValueDbCol++	(Required) Names of the database columns that correspond to ValueAttNames.
WhereClause	(Optional) The WHERE clause that is part of the lookup SELECT query. The lookup query uses this form: <code>SELECT KeyDbCol1, KeyDbCol2, ... , ValueDbCol1, ValueDbCol2, ... FROM Table WHERE WhereClause</code>

Example

The Oracle database table "MyTable = (SYMBOL, ID, PRICE)" is used for lookup. Each data object has four attributes: AttA, AttB, AttC and AttD. AttA and AttB correspond to SYMBOL and ID respectively and are used as a lookup key, and AttD corresponds to PRICE and is added to the data object received from the source. Here is an example of the pipe configuration:

```
adapter.LOOKUPPIPE.ClassName=
com.sybase.esp.adapter.jdbc.JdbcLookupPipe
adapter.LOOKUPPIPE.JdbcUrl = jdbc:oracle:thin:@myhost.com:
1521:mydatabase
adapter.LOOKUPPIPE.JdbcDriver = oracle.jdbc.OracleDriver
adapter.LOOKUPPIPE.DBProperty1 = user
adapter.LOOKUPPIPE.DBProperty1.Value = MyUser
adapter.LOOKUPPIPE.DBProperty2 = password
adapter.LOOKUPPIPE.DBProperty2.Value = MyPassword
adapter.LOOKUPPIPE.Table = MyTable
adapter.LOOKUPPIPE.KeyDbCol1 = SYMBOL
adapter.LOOKUPPIPE.KeyAttName1 = AttA
adapter.LOOKUPPIPE.KeyDbCol2 = ID
```

```

adapter.LOOKUPPIPE.KeyAttName2 = AttB
adapter.LOOKUPPIPE.ValueDbColl = PRICE
adapter.LOOKUPPIPE.ValueAttName1 = AttD
adapter.LOOKUPPIPE.WhereClause = SYMBOL LIKE 'A%'

```

See also

- *Example: Using the JDBCLookupPipe Component on page 293*

Reader Components

The Open adapter has four reader components: MultiFlatXmlStreamReader, XPathXmlStreamReader, XPathMultiTypeXmlReader, and EspDelimitedStreamReader.

MultiFlatXmlStreamReader Properties

The MultiFlatXmlStreamReader component handles messages quickly, provides the flexibility to set defaults based on message content, and splits data into multiple tables in Event Stream Processor.

This reader uses a simple XML format, where the name of the table is the tag and the fields are the attributes.

If MultiFlatXmlStreamReader is selected as the parsing method, sources can populate multiple tables (defined streams). Specify an internal table or tables that Event Stream Processor updates based on data from the source. Also define the fields within each source record by specifying the name and datatype for each field.

Source records for MultiFlatXmlStreamReader have this format:

```
<TableName field1='field1 data' field2='field2 data' ... />
```

Property	Description
AcceptAmper-sand	(Default required) Enter a true or false value to indicate whether the adapter accepts non-XML uses of the ampersand (&). True indicates that the adapter accepts non-XML uses of the ampersand. For example, the adapter converts "&" , "<" , and so on, but it also accepts values such as "Marks & Spencer". False indicates that the adapter rejects non-XML uses of the ampersand.
Type++	(Required) Type the name of the base stream or streams that Event Stream Processor updates based on the data in the source. Repeat this process for each stream you are updating. Ensure each system table has its own Typen property.
Typen . AttName+ +	(Required) Type the name of the table field that Event Stream Processor updates based on the data in the source. This field is case-sensitive. Typen is the name of the related table. Note: Specify a name for each record field in the source data.

Property	Description
<code>TypeN.AtType+</code> +	<p>(Default required) The system defaults the datatype for the field. <code>TypeN</code> is the name of the related table. Event Stream Processor supports these datatypes:</p> <ul style="list-style-type: none"> • <code>string</code> – for strings • <code>datetime</code> – for dates • <code>float</code> – for floating-point numbers • <code>short</code> – for 16-bit signed integers • <code>integer</code> – for 32-bit signed integers • <code>long</code> – for 64-bit signed integers <hr/> <p>Note: Specify a datatype for each record field in the source data.</p>
<code>TypeN.Format++</code>	<p>(Dependent optional) If you created a field with a <code>datetime</code> datatype, enter the format that the adapter understands when reading data for that field. The adapter rejects any data that is not in this format. <code>TypeN</code> is the name of the related table.</p> <p>If you do not specify a value, the adapter understands only <code>datetime</code> values with the format <code>yyyyMMdd</code> or <code>yyyyMMdd HH:mm:ss</code> for the field. It rejects any other <code>datetime</code> data.</p>
<code>TypeN.Match</code>	<p>(Required) Enter the regular expression to match records for this table. <code>TypeN</code> is the name of the related table. Provide a regular expression for each table. For example:</p> <pre>.*table_is_x.*</pre>
<code>TypeN.UTCTime-</code> <code>Zone++</code>	<p>(Dependent optional) If you created a field with a <code>datetime</code> datatype, you can enter the time zone for the field. <code>TypeN</code> is the name of the related table. The adapter converts and normalizes the corresponding <code>datetime</code> value from its originating time zone value to an equivalent UTC value. The UTC value is then passed to Event Stream Processor for storage. You can enter any time zone that Java recognizes (for example, <code>Europe/London</code> or <code>America/New_York</code>).</p> <p>If there is no specified value, the <code>datetime</code> value passes through as local time to Event Stream Processor for storage.</p>

See also

- *Example: Using the MultiFlatXmlStringReader Component* on page 295
- *Valid Time Zones for the Open Adapter* on page 269

XPathXmlStreamReader Properties

The XPathXmlStreamReader component handles XML documents using XPath properties. Select XPathXmlStreamReader as the parsing method to get sources to populate a number of tables.

```
DOSStringReader=com.sybase.esp.adapter.xml.xpath.XPathXmlStreamReader
```

Specify a base stream that Event Stream Processor updates based on data from the source. Also, define the fields within each source record by specifying the name and data type for each field.

You can populate fields with data from an XML document by specifying tag data or attribute values. Specify nested tags by using a forward slash (/) to separate the tag names:

For example, the field data is set to xyz.

```
XPath=/env/body/tag
<env>
<body>
<tag> xyz </tag>
</body>
</env>
```

Attributes are specified by [*@attributeName*].

In this case, the field data is be set to abc.

```
XPath=/env/body/tag[@attr]
<env>
<body>
<tag attr= abc />
</body>
</env>
```

The XPathXmlStreamReader handles collections.

```
XPath=/env/body/tag
<env>
<body>
<tag> xyz </tag>
<tag> abc </tag>
</body>
</env>
```

By default, the command above inserts both values into the field, separated by the collection separator character: xyz|abc.

If a specific tag value is required, use an index operator to specify the position in the collection:

```
XPath=/env/body/tag[ 2 ]
<env>
<body>
<tag> xyz </tag>
```

```
<tag> abc </tag>
</body> </env>
```

This command inserts the value of only the second tag.

Property	Description
XmlRoot	(Required) Enter the root node of the XML document. For example: env
DateFormat	(Optional) If you create fields with a datetime datatype, you can type the default format that the adapter understands when reading data for that field. Unless overridden by the field's Format property, the adapter rejects any data that is not in this format. If you do not enter a value, the adapter only understands datetime values with the format yyyyMMdd or yyyyMMdd HH:mm:ss for the field. It rejects any other datetime data.
Type++	(Required) Name of the base streams that Event Stream Processor updates based on the data in the source.
XPath	(Required) Enter an XPath-style expression for the root node of the table. For example: /env
.AttName++	(Required) Names of the table fields that Event Stream Processor updates based on the data in the source. This property is case-sensitive. Note: Specify a name for each record field in the target Event Stream Processor base stream.
.XPath	(Required) Enter an XPath-style expression for the data to be inserted into this field. If the expression begins with "/", it is taken as a full path. Otherwise, it is relative to the Typen.XPath property. For example: tag or: /env/body/tag Note: If you specify a full path, you cannot access a field at a higher level than the Typen.XPath property.
.DefaultValue	(Optional) If the field is empty, for example, it is an empty tag or the tag is not present in the XML document, this value is substituted.

Property	Description
<code>.Format</code>	(Dependent optional) If you created a field with a datetime datatype, you may type the format that the adapter understands when reading data for that field. The adapter rejects any data that is not in this format. If you do not enter a value, the adapter understands datetime values only with the format <code>yyyyMMdd, yyyyMMdd HH:mm:ss</code> for the field, or with the default value from the <code>DateFormat</code> property. It rejects any other datetime data.
<code>.Match</code>	(Optional) If necessary, type a regular expression match for the adapter to perform on the record. <code>AttName</code> is the field name as defined in the <code>AttName</code> property. If the regular expression is matched in the field data, the string defined in <code>AttName.MatchReplace</code> is substituted. For example: <pre>.*char_is_(.)*</pre>
<code>.MatchReplace</code>	(Dependent optional) Type the replacement value for the <code>.Match</code> property that the adapter may use when the corresponding regular expression match is successful.
<code>.AttType</code>	(Default required) Type the data type for the field. <code>AttName</code> is the field name as defined in the <code>AttName</code> property. Event Stream Processor supports these datatypes: <ul style="list-style-type: none"> • <code>string</code> – for strings • <code>datetime</code> – for dates • <code>float</code> – for floating-point numbers • <code>short</code> – for 16-bit signed integers • <code>integer</code> – for 32-bit signed integers • <code>long</code> – for 64-bit signed integers
<code>.UTCTimeZone</code>	(Dependent optional) If you created a field with a datetime datatype, you may type the time zone for the field. The adapter converts and normalizes the corresponding datetime value from its originating time zone value to an equivalent UTC value. The UTC value is then passed to Event Stream Processor for storage. You may type any time zone that Java recognizes (for example, <code>Europe/London</code> or <code>America/New_York</code>). If no value is set, the datetime value passes through as local time to Event Stream Processor for storage.

Property	Description
BadRecordLoggerName	<p>(Optional) The name of the logger responsible for writing bad records. The behavior depends on implementation.</p> <p>If the name is provided, also provide the implementation class. If this property is left blank, the adapter warns only about bad records but the original message is lost.</p> <pre><BadRecordLoggerName>.ClassName - Logger implementation</pre>

See also

- *Example: Using the XPathXmlStreamReader Component* on page 301
- *Valid Time Zones for the Open Adapter* on page 269

XPathMultiTypeXmlReader Properties

The XPathMultiTypeXmlReader component handles XML messages.

```
DOStringReader=com.sybase.esp.adapter.xml.xpath.XPathMultiTypeXmlReader
```

This reader uses XPathXmlStreamReader, depending on the message type provided in XML. Once the message type is obtained, this component uses the standard XPathXmlStreamReader component to handle incoming messages. All configuration property files for the XPathXmlStreamReader component are stored in separate files called parsing rules. The list of properties in parsing rules are similar to XPathXmlStreamReader except that they require prefix parsing rules.

Property	Description
MSGTypeXPath	(Required) Location of the message type in the XML message expressed as XPath. This can also be provided as an attribute of the element.
MSGTypeN	(Required) Name of message type. One of the message type names must match the value obtained by MSGTypeXPath from XML message.
MSGTypeN.ParsingRules	(Required) Name of the file where XPathXmlStreamReader stores its properties.

Property	Description
BadRecordLoggerName	(Optional) The name of the logger responsible for writing bad records. The behavior depends on implementation. If you provide a name, also provide the implementation class. If you leave this property blank, the adapter only warns about bad records, and the original message is lost. <code><BadRecordLoggerName>.ClassName - Logger implementation.</code>

See also

- *Example: Using the XPathMultiTypeXmlReader Component* on page 300
- *Valid Time Zones for the Open Adapter* on page 269

EspDelimitedStringReader

The EspDelimitedStringReader component is responsible for handling delimited (for example, comma separated) messages. You can use it to send incorrect records to a bad record file.

```
DOStringReader=com.sybase.esp.adapter.dostrings.EspDelimitedStringReader
```

Property	Description
BadRecordLoggerName	(Optional) Name of the logger responsible for writing bad records. The behavior depends on the implementation. If you provide a name, also provide the implementation class. If you leave this property blank, the adapter warns only about bad records, and the original message is lost. <code><BadRecordLoggerName>.ClassName - Logger implementation.</code>
EmptyStringAsNull	(Optional) If set to true, empty string values are translated to null values. For example: NULL If the input record contains: A, B, NULL, D Then the resulting field values is: A, B, {null}, D
FieldDelimiter	(Default required) Character number for a single-character field delimiter. Use either "uXXXX" for a hexadecimal unicode value, or "DDD" for a decimal ASCII value. Use this property if the field is a nonprintable or whitespace character (for example, "space", "tab", or "null"). The default delimiter is the comma (ASCII 44).

Property	Description
NullString	(Optional) A string, which if encountered as a field value, causes the adapter to insert a null string in the resulting message for this field.
StripQuotes	(Optional) If set to true and a field is "quoted", the quote characters are stripped from the beginning and end of the field value. Default value is true.

See also

- *Valid Time Zones for the Open Adapter* on page 269

Writer Component

The Open adapter has one writer component, the XPathXmlWriter.

XPathXmlStringWriter Properties

The XPathXmlWriter component uses an XPath-style syntax to format XML documents from published Event Stream Processor stream data.

The formatter formats XML tags and attributes. To use this writer, ensure the sink specifies this property:

```
DOStringWriter = com.sybase.esp.adapter.xml.xpath.XmlStringWriter
```

Specify nested tags by separating the tag names by / :

```
/env/body/tag
<env>
<body>
<tag>xyz</tag>
</body>
</env>
```

Attributes are specified by [*@attributeName*].

```
/env/body/tag[@attr]
<env>
<body>
<tag attr='xyz' />
</body>
</env>
```

By default, the formatter creates collections. For example, a new nested tag is created for each occurrence of a tag name:

```
XPath1=/env/body/tag
XPath2=/env/body/tag
<env>
<body>
<tag>xyz</tag>
</body>
<body>
<tag>abc</tag>
```

```
</body>
</env>
```

If tags are added within a nesting, use an index operator to specify the position in the collection:

```
XPath1=/env/body[1]/tag
XPath2=/env/body[1]/tag
<env>
<body>
<tag>xyz</tag>
<tag>abc</tag>
</body>
</env>
```

XML content encoding for this component is iso-8859-1.

Property	Description
Type++	(Required) Name of the Event Stream Processor stream to be exported.
.XPath	(Required) The XPath-style description of the top-level tags for this table. <pre>/env/body <env/> <body/></pre>
.AttName++	(Required) Name of a field within the Type (stream).
.XPath	(Required) The XPath-style description of the XML tag or attribute to be formatted. Type is the table name and AttName is the field name specifying the source of the data to insert into the tag or attribute. If the description specifies a tag, the field is output as tag data: <pre>/env/body/tag <env> <body> <tag>field data from Type.AttName</tag> </body> </env></pre> If the description specifies an attribute, the attribute value is set to the field data: <pre>/env/body/tag[@attr] <env> <body> <tag attr= field data from Type.AttName /> </body> </env></pre>

See also

- *Example: Using the XPathXmlStringWriter Component* on page 301

Specifying Datetime Formats

You can specify the acceptable format for dates in a file, if you are using a system that reads data from a file.

The Open adapter rejects dates that are not in the specified format. If you do not specify an acceptable datetime format, the adapter understands datetime values only with the format `yyyyMMdd` or `yyyyMMdd HH:mm:ss`, and rejects any other datetime data.

If you specify the format in the form of a template string, use special identifiers for day, year, month, and so on, along with formatting characters. Use uppercase H for hour to ensure the use of a 24-hour clock.

Table 5. Datetime Format Identifiers

Character	Description	Typical Usage
y	A digit of year	yyyy
M	A digit of month	MM
d	A digit of day	dd
H	A digit of hour (0 – 23)	HH
m	A digit of minute	mm
s	A digit of second	ss
S	A digit of millisecond	SS

If your input data contains letters, enter them in single quotation marks. For example, if the input has strings like `Day:2003-12-29 Time:10:22-00`, specify a datetime format of `'Day':yyyy-MM-dd 'Time':HH:mm:ss`. Entering the format in this way prevents the Open adapter from mistaking the letters as formatting instructions.

Examples of specifying datetime formats:

- If you read dates from a file formatted as `2003/06/29`, where the year is 2003, the month is 06 (June), and the day is 29, enter the datetime format as `yyyy/MM/dd`.
- If you read dates from a file formatted as `29-06-2003 19:12:45`, where the day is 29, the month is 06 (June), the year is 2003, and the time is 7:12:45 PM, enter the datetime format as `dd-MM-yyyy HH:mm:ss`.
- If MQ-Series passes a value for `MQPutDateTime` in the format of `2003-06-29 19:12:45.493`, where the year is 2003, the month is 06 (June), the day is 29, and the time is 7:12:45 PM and 493 milliseconds, enter the datetime format as `yyyy-MM-dd HH:mm:ss.SSS`.

Note: The Open adapter it strips off (Ignores) any milliseconds that it reads through datetime.

See also

- *Valid Time Zones for the Open Adapter* on page 269

Third-Party JAR Files

The Open adapter distribution includes a number of third-party distributable JAR files.

Note: The distribution does not contain the MSSQL JDBC driver, which you can download from <http://www.microsoft.com/downloads>. Search for 'mssql jdbc driver'. The Open adapter supports the SQL Server 2000 driver for JDBC SP3.

File	Description	License Information
.../lib/esp_open_adapter.jar	Contains Sybase Open adapter components	Sybase
.../lib/openadapter.jar	Sybase build of Open adaptor sources based on version 1.7	https://www.openadaptor.org/licence.html
.../jar/esp_java_sdk-0.4.jar	Sybase Event Stream Processor Java SDK library	Sybase
.../lib//jetty/jetty-6.0.1.jar .../lib/jetty/jetty-util-6.0.1.jar .../lib/jetty/servlet-api-2.5-6.0.1.jar	Jetty client libraries	http://www.apache.org/licenses
.../lib/bsh-2.0b4.jar	Library containing BeanShell scripting implementation.	LGPL http://www.beanshell.org/license.html
.../jar/commons-codec-1.3.jar	Part of Apache Commons project	Apache License http://jakarta.apache.org/commons/license.html
.../jar/commons-discovery-0.2.jar	Part of Apache Commons project	Apache License http://jakarta.apache.org/commons/license.html

File	Description	License Information
.../jar/commons-logging-1.1.jar	Part of Apache Commons project	Apache License http://jakarta.apache.org/commons/license.html
.../lib/dom4j-1.5.jar	DOM XML implementation	BSD style http://www.dom4j.org/license.html
.../lib/jakarta-oro-2.0.8.jar	Java classes that provide Perl5 compatible regular expressions, AWK-like regular expressions, glob expressions, and utility classes for performing substitutions, splits, filtering filenames, and so on.	Apache License http://svn.apache.org/repos/asf/jakarta/oro/trunk/LI-CENSE
.../jar/log4j-1.2.14.jar	Logging implementation for Java	Apache License http://logging.apache.org/log4j/docs/index.html
.../jar/xercesImpl.jar	Xerces2 XML implementation	Apache License http://xerces.apache.org/xerces-j/
.../jar/xmlrpc-client-3.1.3.jar .../jar/xmlrpc-common-3.1.3.jar .../jar/xmlrpc-server-3.1.3.jar	XML RPC implementation for Java	Apache License http://ws.apache.org/xmlrpc/
.../jar/adaptapi.jar	External Java adapter framework API	Sybase
.../jar/activation.jar	JavaBeans Activation Framework Specification	SunMicroSystems
.../jar/axis.jar	An implementation of the SOAP submission to W3C	Apache

File	Description	License Information
.../jar/ esp_java_i18n-1.0.jar	Internationalization of messages	Sybase
.../jar/ esp_java_license-1.0.jar	Event Stream Processor licensing API	Sybase
.../jar/mail.jar	Java mail API	Sun Microsystems
.../jar/saa.jar	Web service API	Sun Microsystems
.../jar/sylapi.jar	Required by Sybase Event Stream Processor licensing	Sybase
.../jar/ws-commons-util-1.0.2.jar	Apache Web service common utilities	Apache
.../jar/ wsdl4j-1.5.1.jar	Web services description for Java	Apache

Valid Time Zones for the Open Adapter

Examples of possible valid time zones for `UTCTimeZone` properties in various adapter reader components.

In the `UTCTimeZone` property, you can set any time zone that Java recognizes. Currently, there are over 500 time zones. You can retrieve the comprehensive list in Java through the `TimeZone` object's `getAvailableIDs()` method:

```
TimeZone.getAvailableIDs()
```

See also

- *Specifying Datetime Formats* on page 266
- *MultiFlatXmlStringReader Properties* on page 257
- *XPathXmlStreamReader Properties* on page 259
- *XPathMultiTypeXmlReader Properties* on page 262
- *EspDelimitedStringReader* on page 263

Africa Time Zones

Valid time zones to specify for Africa in the `UTCTimeZone` property.

Country	Time Zone
Algeria	Africa/Algiers
Angola	Africa/Luanda
Benin	Africa/Porto-Novo
Botswana	Africa/Gaborone
Burkina Faso	Africa/Ouagadougou
Burundi	Africa/Bujumbura
Cameroon	Africa/Douala
Cape Verde	Atlantic/Cape_Verde
Central African Republic	Africa/Bangui
Chad	Africa/Ndjamena
Comoros	Indian/Comoro
Democratic Republic of Congo	Africa/Kinshasa Africa/Lubumbashi
Republic of the Congo	Africa/Brazzaville
Cote D'Ivoire	Africa/Abidjan
Djibouti	Africa/Djibouti
Egypt	Africa/Cairo
Equatorial Guinea	Africa/Malabo
Eritrea	Africa/Asmera
Ethiopia	Africa/Addis_Ababa
Gabon	Africa/Libreville
Gambia	Africa/Banjul
Ghana	Africa/Accra
Guinea	Africa/Conakry

CHAPTER 2: Adapters Supported by Event Stream Processor

Country	Time Zone
Guinea-Bissau	Africa/Bissau
Kenya	Africa/Nairobi
Lesotho	Africa/Maseru
Liberia	Africa/Monrovia
Libya	Africa/Tripoli
Madagascar	Indian/Antananarivo
Malawi	Africa/Blantyre
Mali	Africa/Bamako Africa/Timbuktu
Mauritania	Africa/Nouakchott
Mauritius	Indian/Mauritius
Mayotte	Indian/Mayotte
Morocco	Africa/Casablanca
Western Sahara	Africa/El_Aaiun
Mozambique	Africa/Maputo
Namibia	Africa/Windhoek
Niger	Africa/Niamey
Nigeria	Africa/Lagos
Reunion	Indian/Reunion
Rwanda	Africa/Kigali
St Helena	Atlantic/St_Helena
Sao Tome and Principe	Africa/Sao_Tome
Senegal	Africa/Dakar
Seychelles	Indian/Mahe
Sierra Leone	Africa/Freetown
Somalia	Africa/Mogadishu
South Africa	Africa/Johannesburg

CHAPTER 2: Adapters Supported by Event Stream Processor

Country	Time Zone
Sudan	Africa/Khartoum
Swaziland	Africa/Mbabane
Tanzania	Africa/Dar_es_Salaam
Togo	Africa/Lome
Tunisia	Africa/Tunis
Uganda	Africa/Kampala
Zambia	Africa/Lusaka
Zimbabwe	Africa/Harare

Asia Time Zones

Valid time zones to specify for Asia in the `UTCTimeZone` property.

Country	Time Zone
Afghanistan	Asia/Kabul
Armenia	Asia/Yerevan
Azerbaijan	Asia/Baku
Bahrain	Asia/Bahrain
Bangladesh	Asia/Dacca
Bhutan	Asia/Thimbu
British Indian Ocean Territory	Indian/Chagos
Brunei	Asia/Brunei
Burma / Myanmar	Asia/Rangoon
Cambodia	Asia/Phnom_Penh
China	Asia/Harbin Asia/Shanghai Asia/Chungking Asia/Urumqi Asia/Kashgar
Hong Kong	Asia/Hong_Kong

CHAPTER 2: Adapters Supported by Event Stream Processor

Country	Time Zone
Taiwan	Asia/Taipei
Macao	Asia/Macao
Cyprus	Asia/Nicosia
Georgia	Asia/Tbilisi
India	Asia/Calcutta
Indonesia	Asia/Jakarta Asia/Ujung_Pandang Asia/Jayapura
Iran	Asia/Tehran
Iraq	Asia/Baghdad
Israel	Asia/Jerusalem
Japan	Asia/Tokyo
Jordan	Asia/Amman
Kazakhstan	Asia/Almaty Asia/Aqtobe Asia/Aqtau
Kirgizstan	Asia/Bishkek
Korea (North and South)	Asia/Seoul Asia/Pyongyang
Kuwait	Asia/Kuwait Asia/Vientiane
Lebanon	Asia/Beirut
Malaysia	Asia/Kuala_Lumpur Asia/Kuching
Maldives	Indian/Maldives

Country	Time Zone
Mongolia	Asia/Dariv Asia/Ulan_Bator Asia/Baruun-Urt
Nepal	Asia/Katmandu
Oman	Asia/Muscat
Pakistan	Asia/Karachi
Palestine	Asia/Gaza
Philippines	Asia/Manila
Qatar	Asia/Qatar
Saudi Arabia	Asia/Riyadh
Singapore	Asia/Singapore
Sri Lanka	Asia/Colombo
Syria	Asia/Damascus
Tajikistan	Asia/Dushanbe
Thailand	Asia/Bangkok
Turkmenistan	Asia/Ashkhabad
United Arab Emirates	Asia/Dubai
Uzbekistan	Asia/Samarkand Asia/Tashkent
Vietnam	Asia/Saigon
Yemen	Asia/Aden

Australasia Time Zones

Valid time zones to specify for Australasia in the `UTCtimeZone` property.

Country	Time Zone
Australia	Australia/Adelaide Australia/Brisbane Australia/Broken_Hill Australia/Darwin Australia/Hobart Australia/Lindeman Australia/Lord_Howe Australia/Melbourne Australia/Perth Australia/Sydney
Christmas	Indian/Christmas
Cook Islands	Pacific/Rarotonga
Cocos	Indian/Cocos
Fiji	Pacific/Fiji
French Polynesia	Pacific/Gambier Pacific/Marquesas Pacific/Tahiti
Guam	Pacific/Guam
Kiribati	Pacific/Tarawa Pacific/Enderbury Pacific/Kiritimati
North Mariana Islands	Pacific/Saipan
Marshall Island	Pacific/Majuro Pacific/Kwajalein

CHAPTER 2: Adapters Supported by Event Stream Processor

Country	Time Zone
Micronesia	Pacific/Yap Pacific/Truk Pacific/Ponape Pacific/Kosrae
Nauru	Pacific/Nauru
New Caledonia	Pacific/Noumea
New Zealand	Pacific/Auckland Pacific/Chatham
Niue	Pacific/Niue
Norfolk	Pacific/Norfolk
Palau (Belau)	Pacific/Palau
Papua New Guinea	Pacific/Port_Moresby
Pitcairn	Pacific/Pitcairn
American Samoa	American Samoa
W Samoa	Pacific/Apia
Solomon Islands	Pacific/Guadalcanal
Tokelau Islands	Pacific/Fakaofu
Tonga	Pacific/Tongatapu
Tuvalu	Pacific/Funafuti
US minor outlying islands	Pacific/Johnston Pacific/Midway Pacific/Wake
Vanuatu	Pacific/Efate
Wallis and Futuna	Pacific/Wallis

Europe Time Zones

Valid time zones to specify for Europe in the `UTCTimeZone` property.

Country	Time Zone
Andorra	Europe/Andorra
Austria	Europe/Vienna
Belarus	Europe/Minsk
Belgium	Europe/Brussels
Britain / Ireland	Europe/London Europe/Belfast Europe/Dublin
Bulgaria	Europe/Sofia
Czech Republic	Europe/Prague
Denmark, Faeroe Islands, and Greenland	Europe/Copenhagen Atlantic/Faeroe
Estonia	Europe/Tallinn
Finland	Europe/Helsinki
France	Europe/Paris
Germany	Europe/Berlin
Gibraltar	Europe/Gibraltar
Greece	Europe/Athens
Hungary	Europe/Budapest
Iceland	Atlantic/Reykjavik
Italy	Europe/Rome Europe/San_Marino Europe/Vatican
Latvia	Europe/Riga
Liechtenstein	Europe/Vaduz
Lithuania	Europe/Vilnius

CHAPTER 2: Adapters Supported by Event Stream Processor

Country	Time Zone
Luxembourg	Europe/Luxembourg
Malta	Europe/Malta
Moldova	Europe/Chisinau
Monaco	Europe/Monaco
Netherlands	Europe/Amsterdam
Norway	Europe/Oslo
Poland	Europe/Warsaw
Portugal	Europe/Lisbon Atlantic/Azores Atlantic/Madeira
Romania	Europe/Bucharest
Russia	Europe/Kaliningrad Europe/Moscow Europe/Samara Asia/Yekaterinburg Asia/Oms Asia/Krasnoyarsk Asia/Irkutsk Asia/Yakutsk Asia/Vladivostok Asia/Magadan Asia/Kamchatka Asia/Anadyr
Spain	Africa/Ceuta Atlantic/Canary Europe/Madrid
Sweden	Europe/Stockholm
Switzerland	Europe/Zurich

Country	Time Zone
Turkey	Europe/Istanbul Asia/Istanbul
Ukraine	Europe/Kiev Europe/Simferopol
Yugoslavia	Europe/Belgrade Europe/Ljubljana Europe/Sarajevo Europe/Skopje Europe/Zagreb

North America Time Zones

Valid time zones to specify for North America in the `UTCTimeZone` property.

Country	Time Zone
Anguilla	America/Anguilla
Antigua and Barbuda	America/Antigua
Bahamas	America/Nassau
Barbados	America/Barbados
Belize	America/Belize
Bermuda	Atlantic/Bermuda
British Virgin Islands	America/Tortola

CHAPTER 2: Adapters Supported by Event Stream Processor

Country	Time Zone
Canada	America/Dawson America/Dawson_Creek America/Edmonton America/Glace_Bay America/Goose_Bay America/Halifax America/Inuvik America/Iqaluit America/Montreal America/Nipigon America/Pangnirtung America/Rainy_River America/Rankin_Inlet America/Regina America/St_Johns America/Swift_Current America/Thunder_Bay America/Vancouver America/Whitehorse America/Winnipeg America/Yellowknife
Cayman Islands	America/Cayman
Costa Rica	America/Costa_Rica
Cuba	America/Havana
Dominica	America/Dominica
Dominican Republic	America/Santo_Domingo
El Salvador	America/El_Salvador
Grenada	America/Grenada

CHAPTER 2: Adapters Supported by Event Stream Processor

Country	Time Zone
Guadeloupe	America/Guadeloupe
Guatemala	America/Guatemala
Haiti	America/Port-au-Prince
Honduras	America/Tegucigalpa
Jamaica	America/Jamaica
Martinique	America/Martinique
Mexico	America/Cancun America/Chihuahua America/Ensenada America/Mazatlan America/Mexico_City America/Tijuana
Montserrat	America/Montserrat
Nicaragua	America/Managua
Panama	America/Panama
Puerto Rico	America/Puerto_Rico
St Kitts-Nevis	America/St_Kitts
St Lucia	America/St_Lucia
St Pierre and Miquelon	America/Miquelon
St Vincent and the Grenadines	America/St_Vincent
Turks and Caicos	America/Grand_Turk
United States	America/Chicago America/Denver America/Honolulu America/Los_Angeles America/New_York

Country	Time Zone
United States (Alaska)	America/Adak America/Anchorage America/Juneau America/Nome America/Yakutat
United States (exceptions)	America/Boise America/Detroit America/Indiana/Knox America/Indiana/Marengo America/Indiana/Vevay America/Indianapolis America/Louisville America/Menominee America/Phoenix
Virgin Islands	America/St_Thomas

South America Time Zones

Valid time zones to specify for South America in the `UTCTimeZone` property.

Country	Time Zones
Argentina	America/Buenos_Aires America/Catamarca America/Cordoba America/Jujuy America/Mendoza America/Rosario
Aruba	America/Aruba
Bolivia	America/La_Paz

CHAPTER 2: Adapters Supported by Event Stream Processor

Country	Time Zones
Brazil	America/Araguaina America/Belem America/Cuiaba America/Fortaleza America/Maceio America/Manaus America/Noronha America/Porto_Acre America/Porto_Velho America/Sao_Paulo
Chile	America/Santiago Pacific/Easter
Colombia	America/Bogota
Curacao	America/Curacao
Ecuador	America/Guayaquil Pacific/Galapagos
Falklands	Atlantic/Stanley
French Guiana	America/Cayenne
Guyana	America/Guyana
Paraguay	America/Asuncion
Peru	America/Lima
South Georgia	Atlantic/South_Georgia
Suriname	America/Paramaribo
Trinidad and Tobago	America/Port_of_Spain
Uruguay	America/Montevideo
Venezuela	America/Caracas

Starting the Open Adapter

Start an Open adapter instance via a bootstrap class.

Prerequisites

1. Install Java Runtime Environment 1.6.0_26 or higher.
2. Set the JAVA_HOME environment variable to JRE 1.6.0_26 root.

Task

Start the Open adapter using a bootstrap class that reads the configuration file and starts the adapter components:

```
java -Xmx768M -cp ClassPath Bootstrap PropertyFile  
AdapterName
```

where:

- **Xmx768M** – Java parameter specifying the size of the memory heap. You can increase the memory size from 768 for adapter configurations that require more memory.
- **ClassPath** – Java classpath containing the JAR files or classes required by the Open adapter. This includes third-party JAR files. Refer to the component property files within the `examples` directory for more info on the classpath used.
- **Bootstrap** – the adapter bootstrap and a Java class that can be run as a command line program. The name of the class provided is `org.openadapter.adapter.RunAdaptor`.
- **PropertyFile** – name of the properties file containing the component configuration for the adapter `<adapterName>.props`.
- **AdapterName** – name of the adapter to start. When creating adapter configurations, Sybase recommends that you provide adapters with descriptive names to simplify identifying and monitoring adapter processes.

Monitoring the Open Adapter

Monitor a running adapter using RemoteControl and RemoteLogger interfaces.

RemoteControl and RemoteLogger are optional adapter implementations. Specify the interfaces in the adapter properties.

A running adapter contains a controller that supports a dynamic control interface, that can be invoked by a RemoteControl. A RemoteControl provides an interface for operators to communicate with running adapters to establish the current status and resolve problems. To specify a RemoteControl, use this properties syntax:

```
adapterName.Controller.RemoteControl.ClassName = Class
```

Remote logging filters the log output of the adapter and generates alerts or messages. You can configure remote loggers to send all log lines with a specific log level such as FATAL, WARN,

and so on. You can also use a regular expression to pattern match for explicit errors. To specify a RemoteLogger, use this syntax:

```
adapterName. Logging.RemoteLogger.ClassName = Class
```

Sybase provides some standard implementations for remote control and remote logging.

Table 6. Remote Control and Remote Logging Interface Implementations

Interface	Description
HTTPRemote-Control	Turns adapter into a Web server. Point a browser at the machine on which the adapter is running, and the port (default value is 80) on which the HTTPRemoteControl is listening, and HTTPRemoteControl returns a simple control interface.
RvRemote-Control	Allows you to use TIBCO Rendezvous to communicate with the adapter without knowing where the adapter is running.
JMXRemote-Control	A JMX-compliant remote control.
RMIRemote-Control	Registers as an RMI service and allows clients to support, administer, and configure an adapter using RMI.
RvRemoteLogger	Publishes log lines on a specified Rendezvous subject
MailRemote-Logger	Sends e-mail messages using the Java Mail API.

Remote Control Interface

The remote control implementation expects to receive requests as DataObjects that contain specific attributes.

Table 7. Remote Control Interface Attributes

Attribute	Description
Method	Name of the operation.
UserName	Name of the user sending the request.
HostName	Host name the request originates from.
Comment	Any comment.
ControllerName	Name of the Adapter Controller. For example, Adapter.Controller.
Password	If set, ensure this matches the ControlPassword property for the adapter.
Arg1...ArgN	Arguments for the method.

Table 8. Remote Control Methods (Operations)

Operation	Description
pause	Invokes <code>pause()</code> on all adapter components. No messages are processed until the adapter is told to resume.
resume	Invokes <code>resume()</code> on all adapter components. Messages are processed again.
terminate	Invokes <code>terminate()</code> on all adapter components. All source components inform the controller that they are exiting, and the controller then exits.
kill	Invokes <code>System.exit(0)</code> , which ends any controller processes.
logLines	Returns the last N lines from the adapter output logger. The <code>OutputLogger</code> caches the last N lines it writes. The default is 10, but you can change this by using the <code>LogLinesToCache</code> property.
status	Invokes <code>getStatus()</code> , which returns a string on all components and publishes a consolidated status message on the control interface reply subject. The control utility can then display the results. If you write your own component, you can override the <code>getStatus()</code> method.
setLogLevel	Invokes <code>setLogLevel(arg1, arg2)</code> on the adapter <code>OutputLogger</code> . <code>setLogLevel</code> assumes that <code>arg1</code> and <code>arg2</code> in the request <code>DataObject</code> are <code>loglevel</code> and <code>scope</code> . An example is <code>INFO DEFAULT</code> .
customControl	Assumes that <code>arg1</code> in the request <code>DataObject</code> is the name of an adapter component. The remote control forwards the entire request <code>DataObject</code> to the component by calling <code>customControl()</code> on the component. You can edit this.

HTTPRemoteControl

The `HTTPRemoteControl` implementation provides an HTTP-based remote control, turning your adapter into a simple Web server.

The Remote control listens for HTTP requests on a defined port (the default is 80) and replies with a simple HTML interface. This interface represents a control panel, and the buttons generate HTTP get requests. The Remote control parses these URLs into remote control requests.

For adapter A, set:


```
A.Controller.RemoteControl.ClassName =  
org...standard.HTTPRemoteControl  
A.Controller.RemoteControl.HTTPPort = ?  
A.Controller.RemoteControl.ControlPassword = ?
```

The `HTTPPort` and `ControlPassword` properties are optional. They default to port 80 and no password.

To test this remote control, type this URL into a browser: `http://<hostname>:<port>`

You see a control panel that supports the dynamic control interface. The control interface is based on parsing the URL so that you may cut and paste the URLs and use them on existing Web sites. Add `&reply=false` to the URL to disable the control interface in the reply.

The syntax for the URL is:

```
http:// HostName :Port/ ?name= ControllerName &method= Method  
&password= ControlPassword &arg1= Arg-Value ... &argN= ArgValue  
&reply={true|false}
```

The `HTTPRemoteControl` parses this URL and creates a request `DataObject`, which the `AbstractRemoteControl` processes.

MailRemoteLogger

The `MailRemoteLogger` implementation uses the Java Mail API.

The `CLASSPATH` requires `mail.jar` and `activation.jar`. For adapter A, set:

```
A.Logging.RemoteLogSetting = FATAL A.Logging.RemoteLogger.ClassName  
=  
org.openadapter.adapter.mail.MailRemoteLogger  
A.Logging.RemoteLogger.Mailhost = mailhost@foo.com  
A.Logging.RemoteLogger.To = fred@openadaptor.org  
A.Logging.RemoteLogger.FilterPattern = failed to connect
```

Note: `FilterPattern` is an optional property but requires a regular expression. Refer to Java documentation for additional properties.

The standard Open Adapter Controller offers an `OASecurityManager` interface that is responsible for all security related issues. Select an implementation of `OASecurityManager` by setting the controller property `SecurityManager.ClassName` in the controller property file.

PasswordEncryptor

The `PasswordEncryptor` component ensures that there are no plain text passwords in the Open adapter components.

The Event Stream Processor Extension for Open adapter provides sample keystores with the pairs of private and public keys. The default location of keystores is `$ESP_HOME/adapters/esp_open/lib/security`. There are three samples:

CHAPTER 2: Adapters Supported by Event Stream Processor

- `jksKeyStore` – a Java native keystore containing an RSA key pair generated with 512 encryption strength. The password for keystore is "changeit", and the key pair alias is "adaptor".
- `pkcs8KeyStore.der` – a keystore in the form PKCS#8 standard, and encoded using DER. It does not expect a password and alias. It contains an RSA key-pair, and is generated using 512 encryption strength.
- `pkcs12KeyStore.p12` – keystore in the form PKCS#12 standard. The password is "changeit" and alias is "adaptor".

Note: The keystores above are samples only. In a production system, use your own keys.

The Open adapter offers a simple tool to encrypt password strings. In `$ESP_HOME/adapters/esp_open/bin`, the `pwdenc.sh` and `pwdenc.bat` files allow you to encrypt passwords. The tool requires two parameters:

- **-t** – the type of keystore. Valid values are JKS, PKCS8, and PKCS12.
- **-k** – keystore location.

If you provide no settings, the tool uses these default values:

```
pwdenc -t JKS -k ../lib/security/jksKeyStore.der
```

Depending on the keystore type, the tool asks further questions. Encrypted passwords are stored in the `encryptedPwd.txt` file of the directory where the shell script is executed. For example, `$ESP_HOME/adapters/esp_open/bin`. The string is also encoded using base64 algorithm. A limitation is that all characters should be in one line of the adapter property file. Passwords in encrypted form should be copied to the related password field of the component in the adapter property file.

Property	Description
<code>KeyStore</code>	(Required) Location of the Keystore file.
<code>KeyStoreType</code>	(Optional) The standard used to store the Keystore file. Valid values are: JKS, PKCS8, PKCS12. Default value is JKS.
<code>KeyAlias</code>	(Optional) If Keystore type is JKS or PKCS12, provide an alias name for the key pair. This property is not used in PKCS8.
<code>KeyStorePassword</code>	(Optional) If Keystore type is JKS or PKCS12, provide a password. This property is not used in PKCS8.

Generating Self-Signed RSA Keys Using Java Keytool

Use the sample `jksKeyStore` file in the `$ESP_HOME/adapters/esp_open/lib/security` directory to generate self-signed RSA keys using Java keytool.

In a command prompt, execute:

```
keytool -genkey -keyalg rsa -keysize 512 -alias adaptor -keystore jksKeyStore
```

Generating Self-Signed RSA Keys Using OpenSSL

Use the PKCS12 keystore file in the `$ESP_HOME/adapters/esp_open/lib/security` directory to generate self-signed RSA keys using OpenSSL.

1. Generate CA private key.

```
openssl genrsa -rand -des3 -out ca.key 512
```

2. Use that key to create the CA certificate.

```
openssl req -new -x509 -days 365 -key ca.key -out ca.pem -outform PEM
```

3. Export the CA certificate so it can be imported into clientTrustStore.

```
openssl x509 -in ca.pem -out caCert.pem -outform PEM -signkey ca.key
```

4. Generate the server private key.

```
openssl genrsa -rand -des3 -out server.key 512
```

5. Create a server certificate.

```
openssl req -new -days 365 -key server.key -out server.crs
```

6. Sign the server certificate with your CA certificate.

```
openssl ca -in server.crs -out signedServerCert.pem -keyfile ca.key -cert caCert.pem
```

7. Export the certificate to PKCS#12 format so it can be imported to Queue Manager store.

```
openssl pkcs12 -export -in signedServerCert.pem -out pkcs12KeyStore.p12 -inkey server.key -name adaptor
```

Generating Self-Signed RSA Keys Using OpenSSL (PKCS8 Keystore)

Use the sample `pkcs8KeyStore.der` file in the `$ESP_HOME/adapters/esp_open/lib/security` directory to generate self-signed RSA keys using OpenSSL.

In a command prompt, enter:

```
openssl pkcs8 -nocrypt -in server.key -out pkcs8KeyStore.der -outform DER -topk8
```

Examples

See examples of how to operate different Open adapter components.

Example: Using the AsapSink Component

Associate the FilePollSource (reader) component with the AsapSink(writer) component. The FilePollSource component reads records from file on the disk (`insert.txt`) and transfers those records to the AsapSink component. The AsapSink component then publishes those records to Event Stream Processor.

1. Start Event Stream Processor.

Operating System	Step
UNIX	Open a terminal window: <ol style="list-style-type: none"> 1. Start the example cluster: <code>start_server_cluster.sh</code> 2. Start the project on the cluster: <code>start_project.sh</code>
Windows	Open a command window: <ol style="list-style-type: none"> 1. Start the example cluster: <code>start_server_cluster.bat</code> 2. Add project to the cluster, and start it on the cluster: <code>start_project.bat</code>

2. Start **esp_subscriber** to subscribe to the project above that is running on the cluster.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./esp-subscribe.sh</code>
Windows	Open a command window and enter: <code>esp-subscribe.bat</code>

3. Start the FilePollsource and AsapSink components.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./fileToAsap.sh</code>
Windows	Open a command window and enter: <code>fileToAsap.bat</code>

The FilePollsource components reads from the `insert.txt` file and passes records to AsapSink to publish to the Server.

4. In the `fileToAsap.props` file, change `adaptor.FILESOURCE.InputFileName` to `insert_withNULL.txt`, and run again.

See also

- *AsapSink Properties* on page 250

Example: Using the AsapSource Component

Associate the AsapSource (reader) component with the FileSink (writer) component. AsapSource reads records from Event Stream Processor and passes those to FileSink, which then writes those records to the `out.txt` file.

1. Start Event Stream Processor.

Operating System	Step
UNIX	Open a terminal window: <ol style="list-style-type: none"> 1. Start the example cluster: <code>start_server_cluster.sh</code> 2. Start the project on the cluster: <code>start_project.sh</code>
Windows	Open a command window: <ol style="list-style-type: none"> 1. Start the example cluster: <code>start_server_cluster.bat</code> 2. Add project to the cluster, and start it on the cluster: <code>start_project.bat</code>

2. Start `esp_subscriber` to subscribe to the project above that is running on the cluster.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./esp-subscribe.sh</code>
Windows	Open a command window and enter: <code>esp-subscribe.bat</code>

3. Starts the AsapSource and FileSink components.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./asapToFile.sh</code>
Windows	Open a command window and enter: <code>asapToFile.bat</code>

4. Upload data from the `esp_insert.txt` file to the Server.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./esp_upload.sh</code>
Windows	Open a command window and enter: <code>esp_upload.bat</code>

AsapSource reads this published data from the Server and passes it to FileSink, which writes it to the `out.txt` file.

See also

- *AsapSource Properties* on page 246

Example: Using the BeanShellPipe Component

You can use the BeanShellPipe component between the AsapSource and FileSink components. BeanShellPipe executes some commands in shell after it receives data from AsapSource, and before publishing data to FileSink.

1. Start Event Stream Processor.

Operating System	Step
UNIX	Open a terminal window: <ol style="list-style-type: none"> 1. Start the example cluster: <code>start_server_cluster.sh</code> 2. Start the project on the cluster: <code>start_project.sh</code>
Windows	Open a command window: <ol style="list-style-type: none"> 1. Start the example cluster: <code>start_server_cluster.bat</code> 2. Add project to the cluster, and start it on the cluster: <code>start_project.bat</code>

2. Start `esp_subscriber` to subscribe to the project above that is running on the cluster.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./esp-subscribe.sh</code>
Windows	Open a command window and enter: <code>esp-subscribe.bat</code>

3. Start the AsapSource, FileSink and BeanShellPipe components.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./asapToFile.sh</code>
Windows	Open a command window and enter: <code>asapToFile.bat</code>

4. Upload data from the `esp_insert.txt` file to Server.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./esp_upload.sh</code>
Windows	Open a command window and enter: <code>esp_upload.bat</code>

AsapSource reads this data and passes to BeanShellPipe, which then passes it to FileSink, which writes it to the `out.txt` file. BeanShellPipe outputs the text to the command prompt.

See also

- *BeanShellPipe Properties* on page 253

Example: Using the JDBCLookupPipe Component

AsapSource reads data from Event Stream Processor and passes it to the JDBC lookup pipe. If required, the JDBC lookup pipe modifies the values of the 'charfield' column by using 'replaceValue1', and passes that data to FileSource, which then outputs that data to the `file_out.txt` file.

1. Create a table and then create data into the table. For example, for a DB2 database, run the **`createTable_DB2.sql`** script.
Modify this script to use it for any other databases.
2. Update the DB properties in the `JdbcLookupPipe.props` file to point to the required database instance.
3. Update the `JdbcLookupPipe.bat` or `JdbcLookupPipe.sh` script, and add JDBC driver JARs in the classpath.
4. Start Event Stream Processor.

Operating System	Step
UNIX	Open a terminal window: <ol style="list-style-type: none"> 1. Start the example cluster: <code>start_server_cluster.sh</code> 2. Start the project on the cluster: <code>start_project.sh</code>

Operating System	Step
Windows	Open a command window: 1. Start the example cluster: <code>start_server_cluster.bat</code> 2. Add project to the cluster, and start it on the cluster: <code>start_project.bat</code>

5. Start **esp_subscriber** to subscribe to the project above that is running on the cluster.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./esp-subscribe.sh</code>
Windows	Open a command window and enter: <code>esp-subscribe.bat</code>

6. Start the **AsapSource**, **FileSink**, and **JDBClookupPipe** components.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./JdbcLookupPipe.sh</code>
Windows	Open a command window and enter: <code>JdbcLookupPipe.bat</code>

7. Upload data to the the Server.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./esp_upload.sh</code>
Windows	Open a command window and enter: <code>esp_upload.bat</code>

AsapSource reads this data (records) and passes it on to JDBClookupPipe, which modifies the records according to data available and reference data from the database tables. JDBClookupPipe then passes that data to FileSink, which then writes the records to file.

- Table "test1" contains data "col1='AttributeKey'" and "col2='replaceValue1'". 'KeyDbCol1' is col1 in the props file, therefore, col1 column contains attribute keys.

- These attribute keys are present in the incoming record column 'textfield'.
- To replace the 'charfield' column value of a record to 'replaceValue1', then include 'AttributeKey' as a value in 'textfield' column of a record

See the `esp_insert.txt` file for more details. Records that do not have 'AttributeKey' as the 'textfield' column value do not get modified.

8. See contents of the `out.txt` file.

Charfield data for some of the records is updated to 'replaceValue1' value.

See also

- *JDBCLookupPipe Properties* on page 255

Example: Using the MultiFlatXmlStringReader Component

Associate the MultiFlatXmlStringReader component with the FilePollSource component so that it can read records in XML format and pass them to AsapSink, which publishes them to the Server.

1. Start Event Stream Processor.

Operating System	Step
UNIX	Open a terminal window: <ol style="list-style-type: none"> 1. Start the example cluster: <code>start_server_cluster.sh</code> 2. Start the project on the cluster: <code>start_project.sh</code>
Windows	Open a command window: <ol style="list-style-type: none"> 1. Start the example cluster: <code>start_server_cluster.bat</code> 2. Add project to the cluster, and start it on the cluster: <code>start_project.bat</code>

2. Start `esp_subscriber` to subscribe to the project above that is running on the cluster.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./esp-subscribe.sh</code>
Windows	Open a command window and enter: <code>esp-subscribe.bat</code>

3. Start the MultiFlatXmlStringReader, FilePollSource, and AsapSink components.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./MultiFlatXMLStringReader.sh</code>
Windows	Open a command window and enter: <code>MultiFlatXMLStringReader.bat</code>

FilePollSource reads XML formatted data from the `insert.xml` file using MultiFlatXmlStringReader, and passes it to AsapSink, which publishes data to the Server.

See also

- *MultiFlatXmlStringReader Properties* on page 257

Example: Using the SpPersistentSubscribeSource Component

The SpPersistentSubscribeSource component subscribes to the Server using persistent subscribe (stores subscribed records until it processes them, and then deletes them).

To implement this, a log stream (Stream1_log) and truncate stream (TruncateStream1) are created for stream "Stream1". Stream1_log stores the data and TruncateStream1 has two columns, primary key and sequence number. See the `model.ccl` in the `bin` folder for more details.

Incoming records are transferred to Stream1_log with an additional sequencenumber column. Once records are processed from Stream1_log, the last sequence number is published to TruncateStream1. All records with sequence numbers smaller than or equal to the published sequencenumber are then deleted from the Stream1_log.

1. Start Event Stream Processor.

Operating System	Step
UNIX	Open a terminal window: 1. Start the example cluster: <code>start_server_cluster.sh</code> 2. Start the project on the cluster: <code>start_project.sh</code>
Windows	Open a command window: 1. Start the example cluster: <code>start_server_cluster.bat</code> 2. Add project to the cluster, and start it on the cluster: <code>start_project.bat</code>

2. Start `esp_subscriber` to subscribe to Stream1 of the project running on the cluster above.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./esp-subscribe-Stream1.sh</code>
Windows	Open a command window and enter: <code>esp-subscribe-Stream1.bat</code>

3. Subscribe to the log stream, Stream1_Log.

Operating System	Step
UNIX	Open a terminal window and enter: ./esp-subscribe-Stream1_log.sh
Windows	Open a command window and enter: esp-subscribe-Stream1_log.bat

4. Subscribe to the log stream, Truncate_stream1.

Operating System	Step
UNIX	Open a terminal window and enter: ./esp-subscribe-TruncateStream1.sh
Windows	Open a command window and enter: esp-subscribe-TruncateStream1.bat

5. Start the SpPersistentSubscribeSource and FileSink components.

Operating System	Step
UNIX	Open a terminal window and enter: ./asapToFile.sh
Windows	Open a command window and enter: asapToFile.bat

6. Upload data from the `esp_insert.txt` file to the the Server.

Operating System	Step
UNIX	Open a terminal window and enter: ./esp_upload.sh
Windows	Open a command window and enter: esp_upload.bat

SpPersistentSubscribeSource subscribes to the Server and Stream1_log, and passes these records to FileSink, which writes these records to the `out.txt` file. All the subscription script files show the respective subscriptions.

See also

- *SpPersistentSubscribeSource Properties* on page 248

Example: Using the WSSink Component

Use the `WsSink.props` file to associate the `WSSink` component with the `AsapSource` component. `AsapSource` reads data from the Server and passes records to `WSSink`, which publishes these records to a Web service. A second `WsSource.props` file associates the `WSSource` component with `FileSink`. `WSSource` reads published records to the Web service and passes them to `FileSink`, which writes those records to file.

1. Start Event Stream Processor.

Operating System	Step
UNIX	Open a terminal window: <ol style="list-style-type: none"> 1. Start the example cluster: <code>start_server_cluster.sh</code> 2. Start the project on the cluster: <code>start_project.sh</code>
Windows	Open a command window: <ol style="list-style-type: none"> 1. Start the example cluster: <code>start_server_cluster.bat</code> 2. Add project to the cluster, and start it on the cluster: <code>start_project.bat</code>

2. Start `esp_subscriber` to subscribe to the project above that is running on the cluster.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./esp-subscribe.sh</code>
Windows	Open a command window and enter: <code>esp-subscribe.bat</code>

3. Call the `esp_upload` command, and upload records to the Server.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./upload.sh</code>
Windows	Open a command window and enter: <code>upload.bat</code>

4. Start the `WSSource` and `FileSink` components, and the Web service they are connected to.

Operating System	Step
UNIX	Open a terminal window and enter: .WsSource.sh
Windows	Open a command window and enter: WsSource.bat

5. Start the WSSink with AsapSource components.

Operating System	Step
UNIX	Open a terminal window and enter: .WsSink.sh
Windows	Open a command window and enter: WsSink.bat

The `out_wssource.txt` file now contains records. WSSink reads the uploaded records and passes them to the Web service. WSSource reads these records and passes them to FileSink, which writes them to the `out_wssource.txt` file.

See also

- *WSSink Properties* on page 252

Example: Using the WSSource Component

Use the WSSource component to publish data to a Web service using a Web service client, such as soapUI. WSSource reads records from the Web service and passes them to FileSink, which writes those records to file.

1. Start the Web service that WSSource is connected to.

Operating System	Step
UNIX	Open a terminal window and enter: .WsSource.sh
Windows	Open a command window and enter: WsSource.bat

2. Using any SOAP client, try calls given in the `readme.txt` file in WSSource folder, which is located within the `examples` folder. For example, use SOAP client soapUI. This publishes data to a Web service using Web service client. WSSource reads records from the Web service, passes them to FileSink, which writes the records to file.

Example: Using the XPathMultiTypeXmlReader Component

Associate the XPathMultiTypeXmlReader component with the FilePollSource component, which reads XML formatted data, and with the AsapSink component, which publishes records to the Server. Define parsing rules in the XPathXmlStreamReader.props file, and use these rules to parse XML records being read from file.

1. Start Event Stream Processor.

Operating System	Step
UNIX	Open a terminal window: 1. Start the example cluster: <code>start_server_cluster.sh</code> 2. Start the project on the cluster: <code>start_project.sh</code>
Windows	Open a command window: 1. Start the example cluster: <code>start_server_cluster.bat</code> 2. Add project to the cluster, and start it on the cluster: <code>start_project.bat</code>

2. Start `esp_subscriber` to subscribe to the project above that is running on the cluster.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./esp-subscribe.sh</code>
Windows	Open a command window and enter: <code>esp-subscribe.bat</code>

3. Start the FilePollSource (with XPathXmlStreamReader) and AsapSink components.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./XPathMultiTypeXmlReader.sh</code>
Windows	Open a command window and enter: <code>XPathMultiTypeXmlReader.bat</code>

Data from the `insert.xml` file publishes to the Server.

See also

- *XPathMultiTypeXmlReader Properties* on page 262

Example: Using the XPathXmlStreamReader Component

Use XPathXmlStreamReader with FilePollSource, which reads XML data and parses it using XPath rules. Then pass the records to AsapSink, which publishes them to the Server.

1. Start Event Stream Processor.

Operating System	Step
UNIX	Open a terminal window: <ol style="list-style-type: none"> 1. Start the example cluster: <code>start_server_cluster.sh</code> 2. Start the project on the cluster: <code>start_project.sh</code>
Windows	Open a command window: <ol style="list-style-type: none"> 1. Start the example cluster: <code>start_server_cluster.bat</code> 2. Add project to the cluster, and start it on the cluster: <code>start_project.bat</code>

2. Start `esp_subscriber` to subscribe to the project above that is running on the cluster.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./esp-subscribe.sh</code>
Windows	Open a command window and enter: <code>esp-subscribe.bat</code>

3. Start the FilePollsource (with XPathXmlStreamReader) and AsapSink components.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./XPathXmlStreamReader.sh</code>
Windows	Open a command window and enter: <code>XPathXmlStreamReader.bat</code>

Data from the `insert.xml` file publishes to the Server.

See also

- *XPathXmlStreamReader Properties* on page 259

Example: Using the XPathXmlStringWriter Component

The XPathXmlStringWriter component writes XML formatted data using XPath rules, and is used with the FileSink component. Associate the AsapSource component, which reads data

CHAPTER 2: Adapters Supported by Event Stream Processor

from the Server and passes it to FileSink, with the FileSink component, which writes out the XML data using XPathXmlStringWriter.

1. Start Event Stream Processor.

Operating System	Step
UNIX	Open a terminal window: 1. Start the example cluster: <code>start_server_cluster.sh</code> 2. Start the project on the cluster: <code>start_project.sh</code>
Windows	Open a command window: 1. Start the example cluster: <code>start_server_cluster.bat</code> 2. Add project to the cluster, and start it on the cluster: <code>start_project.bat</code>

2. Start `esp_subscriber` to subscribe to the project above that is running on the cluster.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./esp-subscribe.sh</code>
Windows	Open a command window and enter: <code>esp-subscribe.bat</code>

3. Start the AsapSource and FileSink (with XPathXmlStringWriter) components.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./XPathXmlStringWriter.sh</code>
Windows	Open a command window and enter: <code>XPathXmlStringWriter.bat</code>

Data from the `insert.xml` file publishes to the Server.

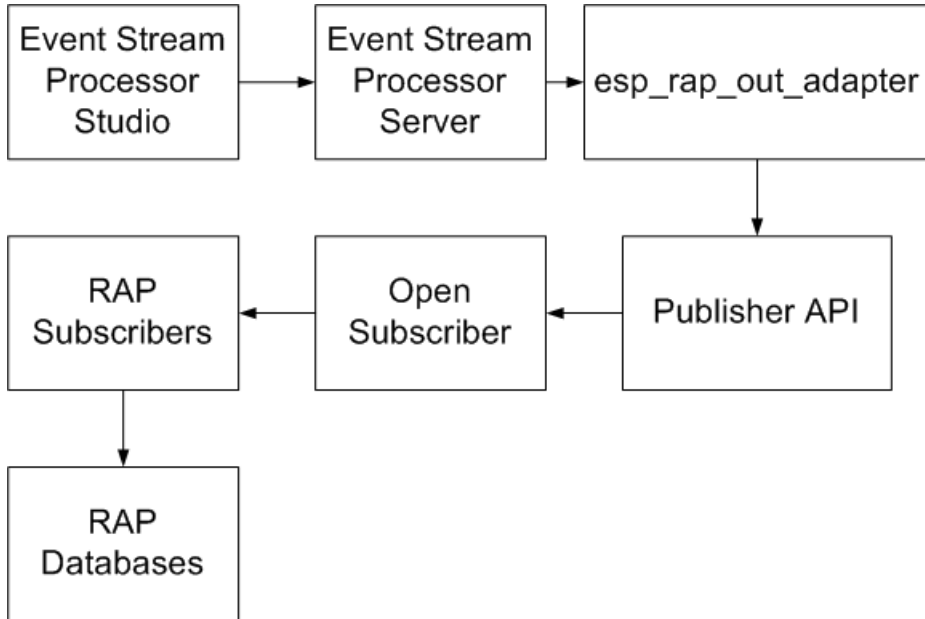
See also

- *XPathXmlStringWriter Properties* on page 264

RAP Adapter

Adapter type: `esp_rap_out_adapter`. The Sybase Event Stream Processor RAP adapter is an external adapter that publishes data from the Event Stream Processor to the RAP platform using the C SDK.

The RAP adapter supports only Solaris and Linux platforms.



Each stream you want to publish to the RAP platform requires its own adapter. For example, to publish three streams to RAP, configure three adapters. Start and stop these adapters separately.

Note: Since RAP accepts only inserts as input, deletes from the Server are dropped, and updates are converted to inserts.

Start Command

Use the `start.sh` script to publish data from the Server to RAP.

Syntax

To use the `start.sh` script, copy the platform specific `libpublisher.so` to the `$ESP_HOME/adapters/rap_out/lib` directory.

Note: The `start.sh` is an implementation of the `esp_rap_out_adapter` command.

```

esp_rap_out_adapter -f configFile -t templateDir -p
publisherConfigDir &
  
```

Note: Ensure that `libodbc.so` is installed if it is not present. A symbolic link with the file name `libodbc.so.1` should be made to `libodbc.so` version 1.0.0 and this file should be put in `$ESP_HOME/adapters/rap_out/lib`.

Required Arguments

Argument	Description
<code>-f configFile</code>	The full path of the configuration file that specifies the streams from Event Stream Processor that provide data to RAP. Default value is <code>../config/espfeedhandler.xml</code>
<code>-t templateDir</code>	The full path to the directory containing the RDS templates that map columns in the Event Stream Processor streams to tables and columns of RAP. Default value is <code>../templates</code>
<code>-p PublisherConfigDir</code>	The full path to the directory containing the publisher configuration file, which defines the multicast address used by the RAP subscriber. Default value is <code>../config</code>
<code>&</code>	(Optional) Run the adapter in the background.

See also

- *Starting the RAP Adapter* on page 319

Stop Command

If the RAP adapter is running in the foreground, you can stop it by pressing Ctrl+C.

If the RAP adapter is running in the background, you can stop it by entering `ps -eaf | grep esp_rap_out` to get the process ID of the adapter and enter `kill -ll processID` in the command line.

See also

- *Stopping the RAP Adapter* on page 319

Datatype Mapping for the RAP Adapter

Event Stream Processor datatypes map to RAP, ASE, and IQ datatypes.

The RAP adapter does not support the Event Stream Processor binary and boolean datatypes.

CHAPTER 2: Adapters Supported by Event Stream Processor

ESP Datatype	RAP Datatype	ASE Datatype	IQ Datatype
integer	sint32	int	int
long	sint64	bigint	bigint
float	decimal(p,s)	decimal(p,s) or numeric(p,s)	decimal(p,s) or numeric(p,s)
interval	sint64	bigint	bigint
date	datetime	datetime	timestamp
timestamp	datetime	datetime	timestamp
bigdatetime	datetime2	bigdatetime	timestamp
money	decimal(19,4)	numeric(19,4)	numeric(19,4)
money(1)	decimal(19,1)	decimal(19,1)	decimal(19,1)
money(2)	decimal(19,2)	decimal(19,2)	decimal(19,2)
money(3)	decimal(19,3)	decimal(19,3)	decimal(19,3)
money(4)	decimal(19,4)	decimal(19,4)	decimal(19,4)
money(5)	decimal(19,5)	decimal(19,5)	decimal(19,5)
money(6)	decimal(19,6)	decimal(19,6)	decimal(19,6)
money(7)	decimal(19,7)	decimal(19,7)	decimal(19,7)
money(8)	decimal(19,8)	decimal(19,8)	decimal(19,8)
money(9)	decimal(19,9)	decimal(19,9)	decimal(19,9)
money(10)	decimal(19,10)	decimal(19,10)	decimal(19,10)
money(11)	decimal(19,11)	decimal(19,11)	decimal(19,11)
money(12)	decimal(19,12)	decimal(19,12)	decimal(19,12)
money(13)	decimal(19,13)	decimal(19,13)	decimal(19,13)
money(14)	decimal(19,14)	decimal(19,14)	decimal(19,14)
money(15)	decimal(19,15)	decimal(19,15)	decimal(19,15)
string	string	varchar(n)	varchar(n)

Configuration

Configuration information for the RAP adapter.

To configure the RAP adapter, you need:

- An adapter configuration file
- A publisher file
- An RDS template file

Adapter Configuration File

Use the `espsfeedhandler.xml` configuration file to specify which Event Stream Processor streams provide data to RAP.

Syntax

```
<?xml version="1.0" encoding="UTF-8"?>
<ESPFeedHandler>
  <Logger>
    <LogLevel>warning</LogLevel>
    <LogFile>ESPFeedHandler.log</LogFile>
  </Logger>
  <MainCommandControlServer>
  <MainCCHost>127.0.0.1</MainCCHost>
  <MainCCPort>55555</MainCCPort>
  <Workspace>workspace1</Workspace>
  <Project>project1</Project>
  </MainCommandControlServer>
  <StandbyCommandControlServer>
  <StandbyCCHost/>
  <StandbyCCPort/>
  </StandbyCommandControlServer>
  <UseEncryption/>
  <ESPAuthentication>
  <User></User>
  <Password></Password>
  </ESPAuthentication>
  <Subscription>          <ProjectionSQL></ProjectionSQL>
<SubscriptionStream>ds1</SubscriptionStream>
  <RAPMessageType>69</RAPMessageType>
  </Subscription>
</ESPFeedHandler>
```

Table 9. XML Elements

Element	Description
ESPFeedHandler	(Required) The root element of the file.
Logger	(Required) The root element for logging activities settings.

Element	Description
LogLevel	(Required) The level of logging. Valid values are: <ul style="list-style-type: none"> error – logs only errors. warning – logs warnings and errors. info – logs informational messages and messages logged at the warning level. debug – logs debugging messages and messages logged at the info level.
LogFile	(Required) The name and location (relative or absolute path) of the log file.
MainCommandControlServer	The root element of the connection information for the main command control server.
MainCCHost	(Required) The IP address of the main command control server.
MainCCPort	(Required) The port for the main command control server.
Workspace	(Required) The workspace in the cluster that contains the stream.
Project	(Required) The project that contains the stream.
StandbyCommandControlServer	(Optional) The root element of the connection information for the standby command control server.
StandbyCCHost	(Required) The IP address of the standby command control server.
StandbyCCPort	(Required) The port for the standby command control server.
UseEncryption	(Optional) The root element for SSL encryption for communications between the RAP adapter and Event Stream Processor.
ESPAuthentication	(Required) The root element for the information necessary for authenticating the connection to the command control server.
User	(Required) The user name to connect to the command control server.
Password	(Required) The password to use to connect to the command control server (in encrypted form).
Subscription	(Required) The root element for information about subscription to a stream.
ProjectionSQL	(Optional) The SQL query projection to be used on the stream.
SubscriptionStream	(Optional) The name of the stream to which you want to subscribe.

Element	Description
RAPMessageType	(Required) The RAP message type number. This is the same number used in the RDS template.

Publisher File

The publisher.xml file configures the RAP publisher in Event Stream Processor. It specifies the log file, administration channel, and data stream channels.

For more information about how to configure the RAP publisher, see *Configuring a Publisher* in the *RAP - The Trading Edition R4.1 Operations Console Users Guide*.

Syntax

```
<?xml version="1.0" encoding="UTF-8"?>
<Publisher>
  <Logger>
    <LogLevel>...</LogLevel>
    <LogFile>...</LogFile>
  </Logger>
  <NumMessageBuffers>...</NumMessageBuffers>
  <NumPacketBuffers>...</NumPacketBuffers>
  <MessageFlushInterval>...</MessageFlushInterval>
  <LatencyCheckInterval>...</LatencyCheckInterval>
  <AdminChannel>
    <LocalInterface>...</LocalInterface>
    <AdminPort>...</AdminPort>
    <MaxConnections>...</MaxConnections>
  </AdminChannel>
  <ResendChannel>
    <ResendPort>...</ResendPort>
  </ResendChannel>
  <TimeToLive>...</TimeToLive>
  <DataStreamChannelList>
    <DataStreamChannel>
      <ChannelName>...</ChannelName>
      <LocalInterface>...</LocalInterface>
      <IPAddress>...</IPAddress>
      <Port>...</Port>
    </DataStreamChannel>
    <DataStreamChannel>
      <ChannelName>...</ChannelName>
      <LocalInterface>...</LocalInterface>
      <IPAddress>...</IPAddress>
      <Port>...</Port>
    </DataStreamChannel>
  </DataStreamChannelList>
</Publisher>
```

Table 10. XML Elements

Element	Description
Publisher	Root element for the configuration file.
Logger	Contains settings for logging activities.
LogLevel	The level of logging. Valid values are: <ul style="list-style-type: none"> • Error – log only errors. • Warning – log warnings in addition to errors. • Info – log informational messages in addition to messages logged at the warning level. • Debug – log debugging messages in addition to messages logged at the info level.
LogFile	Name and location of the log file. The file name can be relative or a full path.
NumMessageBuffers	Number of message buffers. One message buffer is required for each message that is being simultaneously built. This setting can have a value from 1 to 65535, although the machine must have enough memory to hold the number of buffers specified.
NumPacketBuffers	Maximum number of packets to cache to satisfy requests by a subscriber to resend a packet. The number of packets is cached per data stream channel. The setting can have a value from 1 to 4294967296, although the machine must have enough memory to hold the number of packets specified. The number of buffers is allocated on initialization of the publisher.
MessageFlushInterval	Interval, in seconds, during which a partially filled message buffer must be idle before being sent on the network. This setting can have a value from 1 to 65535.
LatencyCheckInterval	Number of seconds after which to perform a latency check on a message. This setting can have a value from 1 to 65535.
AdminChannel	Information about the administration channel. This channel accepts requests for version information, statistics, and shutdown.
LocalInterface	Local interface that the publisher uses to monitor administrative requests.
AdminPort	Port number used by the UAF agent to communicate with the publisher. The publisher listens for incoming administration requests on this port.

Element	Description
MaxConnections	Determines the number of concurrent connections to the Admin-Channel. Value must be in the range of 1 to 65535. Default value is 10
ResendChannel	Information about the resend channel. This channel listens for connections from subscribers, who open connections to publishers and issue requests to resend packets.
ResendPort	Port number used by subscribers to request resends of dropped network packets, and to time network latency between publisher and subscriber.
TimeToLive	The limit on the number of routing devices a message may pass through before expiring.
DataStreamChannelList	A list of data stream channel definitions. There can be up to 255 data stream channels.
DataStreamChannel	Contains information for one data stream channel. Each message sent by the publisher is sent in a network packet buffer over one of the defined channels. The publisher attempts to balance sending over all the channels while the system is under load.
ChannelName	Descriptive name for the channel, which is used to identify the channel when logging.
LocalInterface	IP address of a network interface on the local machine, which should be used for sending data.
IPAddress	UDP multicast address for sending the messages over the network.
Port	Port over which messages are sent using UDP multicasting.

RDS Template File

The RAP data stream (RDS) template file defines the structure of RAP message types in Event Stream Processor.

For more information on RAP Messages and Schemas, see *Customizing the RAP Messages and Schema* Chapter in the *RAP - The Trading Edition R4.1 Developers Guide*.

Syntax

```
<?xml version="1.0" encoding="UTF-8"?>

<Template xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="../template.xsd">
  <MessageDefnList>
    <MessageDefn>
```



```

<MessageDesc>...</MessageDesc>
<MessageType>...</MessageType>
<DestTableName>...</DestTableName>
<FieldDefnList>
  <FieldDefn>
    <FieldName>...</FieldName>
    <StringField/>
    <DestColumnName>...</DestColumnName>
    <Lookup>
      <LookupTableName>...T</LookupTableName>
      <LookupColumnName>...</LookupColumnName>
      <LookupColumnReturn>...</LookupColumnReturn>
    </Lookup>
  </FieldDefn>
</FieldDefnList>
</MessageDefn>
</MessageDefnList>
</Template>

```

Table 11. XML Elements

Element	Description
Template	Root element for the template.
MessageDefnList	A list of one or more message definitions.
MessageDefn	Information that defines a single message type.
MessageDesc	Description of the type of market data message, for example, Stock Quote. This element is used only for descriptive purposes, and can contain any string.
MessageType	Unique number representing the type of market data message. This number must uniquely identify the message type across all message definitions within all templates. The value can contain any integer from 1 – 65535.
DestTableName	Name of the database table into which the message should be stored. There is one database table per message type. The value can contain any string.
FieldDefnList	A list of one or more field definitions.
FieldDefn	Information that defines a single field.
FieldName	Name of the field. The value can contain any string.
IntegerField	Indicates that the field is some type of integer datatype.
IntegerDataType	Datatype of an integer field. Valid values: uint8, uint16, uint32, uint64, sint8, sint16, sint32, or sint64.

Element	Description
DecimalField	Indicates that the field is a decimal value. The field definition can contain only one of: IntegerField, DecimalField, StringField, DateField, TimeField, DateTimeField, DateTime2Field, or Time2Field.
Precision	Precision of a decimal field. Maximum precision allowed is 38 digits.
Scale	Scale of a decimal field (the number of digits after the decimal point). Maximum scale can be no larger than the precision (38).
StringField	Indicates that the field is a string datatype.
DateField	Indicates that the field is a date datatype. The field definition can contain only one of: IntegerField, DecimalField, StringField, DateField, TimeField, DateTimeField, DateTime2Field, or Time2Field.
TimeField	Indicates that the field is a time datatype.
Time2Field	Indicates that the field is a bigtime datatype (granularity to 6 decimal places). The field definition can contain only one of: IntegerField, DecimalField, StringField, DateField, TimeField, DateTimeField, DateTime2Field, or Time2Field.
DateTimeField	Indicates that the field is a datetime datatype.
DateTime2Field	Indicates that the field is a bigdatetime (granularity to 6 decimal places). The field definition can contain only one of: IntegerField, DecimalField, StringField, DateField, TimeField, DateTimeField, DateTime2Field, or Time2Field.
DestColumnName	Name of the column into which the field data should be stored. There is one column per field. The value of this element can contain any string.
Lookup	(Optional) Indicates that the data in the field should be used as a lookup for another table. If this element does not appear in a field definition, then no lookup is required.
LookupTableName	Name of the table to use to look up a value.
LookupColumnName	Name of the column to use to look up a value.
LookupColumnReturn	Name of the column from which to return data when doing a lookup.

Example: Configuring the RAP Adapter

Set the configuration, publisher, and RDS template files to configure the RAP adapter for communication between RAP and Event Stream Processor.

1. Set the \$RAPOUT_HOME environment variable to \$ESP_HOME/adapters/rap_out directory.
2. Navigate to the \$RAPOUT_HOME directory.
3. Create a project defining the streams you want to publish to RAP, and save it to a file named model.ccl in the \$ESP_HOME/bin directory.
4. Start Event Stream Processor.

Windows:

1. Start the example cluster.

```
cd %ESP_HOME%\cluster\nodes\node1
%ESP_HOME%\bin\esp_server --cluster-node node1.xml
```

2. Compile CCL to create CCX.

```
%ESP_HOME%\bin\esp_compiler -i model.ccl -o model.ccx
```

3. Deploy the project on the cluster.

```
%ESP_HOME%\bin\esp_cluster_admin" --uri=esp://localhost:19011
--username=sybase --password=sybase --add_project --workspace-
name=w1 --project-name=p1 --ccx=model.ccx
```

4. Start the deployed project on the cluster.

```
%ESP_HOME%\bin\esp_cluster_admin" --uri=esp://localhost:19011
--username=sybase --password=sybase --start_project --
workspace-name=w1 --project-name=p1
```

UNIX:

1. Start the example cluster.

```
cd $ESP_HOME/cluster/nodes/node1
$ESP_HOME/bin/esp_server --cluster-node node1.xml
```

2. Compile CCL to create CCX.

```
$ESP_HOME/bin/esp_compiler -i model.ccl -o model.ccx
```

3. Deploy the project on the cluster.

```
$ESP_HOME/bin/esp_cluster_admin" --uri=esp://localhost:19011
--username=sybase --password=sybase --add_project --workspace-
name=w1 --project-name=p1 --ccx=model.ccx
```

4. Start the deployed project on the cluster.

```
$ESP_HOME/bin/esp_cluster_admin" --uri=esp://localhost:19011
--username=sybase --password=sybase --start_project --
workspace-name=w1 --project-name=p1
```

5. In the \$RAPOUT_HOME/config directory, modify the espfeedhandler.xml configuration file to specify which Event Stream Processor streams are providing data to RAP. For example, the file below configures the adapter to publish a single stream called Trades:

```
<?xml version="1.0" encoding="UTF-8"?>
<ESPFeedHandler>
```

```

    <Logger>
      <LogLevel>warning</LogLevel>
      <LogFile>ESPFeedHandler.log</LogFile>
    </Logger>
    <MainCommandControlServer>
      <MainCCHost>127.0.0.1</MainCCHost>
      <MainCCPort>19011</MainCCPort>
      <Workspace>wl</Workspace>
      <Project>p1</Project>
    </MainCommandControlServer>
    <StandbyCommandControlServer>
      <StandbyCCHost/>
      <StandbyCCPort/>
    </StandbyCommandControlServer>
    <UseEncryption/>
    <ESPAuthentication>
      <User></User>
      <Password></Password>
    </ESPAuthentication>
    <Subscription>
      <ProjectionSQL></ProjectionSQL>
      <SubscriptionStream>Trades</SubscriptionStream>
      <RAPMessageType>69</RAPMessageType>
    </Subscription>
  </ESPFeedHandler>

```

6. In `$RAPOUT_HOME/config`, modify the existing publisher file to specify the multicast address used by the RAP subscriber. For example:

```

<?xml version="1.0" encoding="UTF-8"?>
<Publisher>
  <Logger>
    <LogLevel>debug</LogLevel>
    <LogFile>Publisher.log</LogFile>
  </Logger>
  <NumMessageBuffers>1</NumMessageBuffers>
  <NumPacketBuffers>10000</NumPacketBuffers>
  <MessageFlushInterval>1</MessageFlushInterval>
  <LatencyCheckInterval>30</LatencyCheckInterval>
  <AdminChannel>
    <LocalInterface>testmachine</LocalInterface>
    <AdminPort>5002</AdminPort>
  </AdminChannel>
  <ResendChannel>
    <ResendPort>5103</ResendPort>
  </ResendChannel>

  <TimeToLive>1</TimeToLive>

  <DataStreamChannelList>
    <DataStreamChannel>
      <ChannelName>test2</ChannelName>
      <LocalInterface>127.0.0.1</LocalInterface>
      <IPAddress>224.0.2.0</IPAddress>
      <Port>5050</Port>
    </DataStreamChannel>
  </DataStreamChannelList>

```

```

    <DataStreamChannel>
      <ChannelName>test1</ChannelName>
      <LocalInterface>127.0.0.1</LocalInterface>
      <IPAddress>224.0.2.0</IPAddress>
      <Port>5800</Port>
    </DataStreamChannel>
  </DataStreamChannelList>
</Publisher>

```

7. In \$RAPOUT_HOME/templates, create an RDS template for each stream you want to publish to RAP.

```

<?xml version="1.0" encoding="UTF-8"?>

<Template xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="../template.xsd">
  <MessageDefnList>
    <MessageDefn>
      <MessageDesc>Split Event</MessageDesc>
      <MessageType>70</MessageType>
      <DestTableName>rapout2</DestTableName>
      <FieldDefnList>
        <FieldDefn>
          <FieldName>integer</FieldName>
          <IntegerField>
            <IntegerDataType>sint32</IntegerDataType>
          </IntegerField>
          <DestColumnName>int16</DestColumnName>
        </FieldDefn>
        <FieldDefn>
          <FieldName>string</FieldName>
          <StringField/>
          <DestColumnName>string</DestColumnName>
        </FieldDefn>
        <FieldDefn>
          <FieldName>int32</FieldName>
          <IntegerField>
            <IntegerDataType>sint32</IntegerDataType>
          </IntegerField>
          <DestColumnName>int32test</DestColumnName>
        </FieldDefn>
        <FieldDefn>
          <FieldName>int64</FieldName>
          <IntegerField>
            <IntegerDataType>sint64</IntegerDataType>
          </IntegerField>
          <DestColumnName>int64test</DestColumnName>
        </FieldDefn>
        <FieldDefn>
          <FieldName>double</FieldName>
          <DecimalField>
            <Precision>18</Precision>
            <Scale>4</Scale>
          </DecimalField>
          <DestColumnName>doubletest</DestColumnName>
        </FieldDefn>

```

```

<FieldDefn>
  <FieldName>money</FieldName>
  <DecimalField>
    <Precision>18</Precision>
    <Scale>4</Scale>
  </DecimalField>
  <DestColumnName>money_test</DestColumnName>
</FieldDefn>
<FieldDefn>
  <FieldName>money(1)</FieldName>
  <DecimalField>
    <Precision>18</Precision>
    <Scale>1</Scale>
  </DecimalField>
  <DestColumnName>money1</DestColumnName>
</FieldDefn>
<FieldDefn>
  <FieldName>money(2)</FieldName>
  <DecimalField>
    <Precision>18</Precision>
    <Scale>2</Scale>
  </DecimalField>
  <DestColumnName>money2</DestColumnName>
</FieldDefn>
<FieldDefn>
  <FieldName>money(3)</FieldName>
  <DecimalField>
    <Precision>18</Precision>
    <Scale>3</Scale>
  </DecimalField>
  <DestColumnName>money3</DestColumnName>
</FieldDefn>
<FieldDefn>
  <FieldName>money(4)</FieldName>
  <DecimalField>
    <Precision>18</Precision>
    <Scale>4</Scale>
  </DecimalField>
  <DestColumnName>money4</DestColumnName>
</FieldDefn>
<FieldDefn>
  <FieldName>money(5)</FieldName>
  <DecimalField>
    <Precision>18</Precision>
    <Scale>5</Scale>
  </DecimalField>
  <DestColumnName>money5</DestColumnName>
</FieldDefn>
<FieldDefn>
  <FieldName>money(6)</FieldName>
  <DecimalField>
    <Precision>18</Precision>
    <Scale>6</Scale>
  </DecimalField>
  <DestColumnName>money6</DestColumnName>
</FieldDefn>

```

```

<FieldDefn>
  <FieldName>money(7)</FieldName>
  <DecimalField>
    <Precision>18</Precision>
    <Scale>7</Scale>
  </DecimalField>
  <DestColumnName>money7</DestColumnName>
</FieldDefn>
<FieldDefn>
  <FieldName>money(8)</FieldName>
  <DecimalField>
    <Precision>18</Precision>
    <Scale>8</Scale>
  </DecimalField>
  <DestColumnName>money8</DestColumnName>
</FieldDefn>
<FieldDefn>
  <FieldName>money(9)</FieldName>
  <DecimalField>
    <Precision>18</Precision>
    <Scale>9</Scale>
  </DecimalField>
  <DestColumnName>money9</DestColumnName>
</FieldDefn>
<FieldDefn>
  <FieldName>money(10)</FieldName>
  <DecimalField>
    <Precision>18</Precision>
    <Scale>10</Scale>
  </DecimalField>
  <DestColumnName>money10</DestColumnName>
</FieldDefn>
<FieldDefn>
  <FieldName>money(11)</FieldName>
  <DecimalField>
    <Precision>18</Precision>
    <Scale>11</Scale>
  </DecimalField>
  <DestColumnName>money11</DestColumnName>
</FieldDefn>
<FieldDefn>
  <FieldName>money(12)</FieldName>
  <DecimalField>
    <Precision>18</Precision>
    <Scale>12</Scale>
  </DecimalField>
  <DestColumnName>money12</DestColumnName>
</FieldDefn>
<FieldDefn>
  <FieldName>money(13)</FieldName>
  <DecimalField>
    <Precision>18</Precision>
    <Scale>13</Scale>
  </DecimalField>
  <DestColumnName>money13</DestColumnName>
</FieldDefn>

```

```

<FieldDefn>
  <FieldName>money(14)</FieldName>
  <DecimalField>
    <Precision>18</Precision>
    <Scale>14</Scale>
  </DecimalField>
  <DestColumnName>money14</DestColumnName>
</FieldDefn>
<FieldDefn>
  <FieldName>money(15)</FieldName>
  <DecimalField>
    <Precision>18</Precision>
    <Scale>15</Scale>
  </DecimalField>
  <DestColumnName>money15</DestColumnName>
</FieldDefn>
<FieldDefn>
  <FieldName>interval</FieldName>
  <IntegerField>
    <IntegerDataType>sint64</IntegerDataType>
  </IntegerField>
  <DestColumnName>interval</DestColumnName>
</FieldDefn>
<FieldDefn>
  <FieldName>bigdatettime</FieldName>
  <DateTime2Field/>
  <DestColumnName>bigdatettime</DestColumnName>
</FieldDefn>
<FieldDefn>
  <FieldName>date</FieldName>
  <DateTimeField/>
  <DestColumnName>date_test</DestColumnName>
</FieldDefn>
<FieldDefn>
  <FieldName>timestamp</FieldName>
  <DateTimeField/>
  <DestColumnName>timestamp_test</DestColumnName>
</FieldDefn>
</FieldDefnList>
</MessageDefn>
</MessageDefnList>
</Template>

```

Ensure that the template file is copied to the RAP subscriber template directory

Operation

Start and stop the RAP adapter from the command line.

Starting the RAP Adapter

Once you have configured the adapter, start it using the **start.sh** script.

Prerequisites

- Start the RAP databases (there are message tables in the database), RAP subscribers, the Server, and that the project you want the adapter to connect to.
- Install `libodbc.so` if it is not present. A symbolic link with the file name `libodbc.so.1` should be made to `libodbc.so` version 1.0.0 and this file should be put in `$ESP_HOME/adapters/rap_out/lib`.
- To use the `start.sh` script, copy the platform specific `libpublisher.so` to the `$ESP_HOME/adapters/rap_out/lib` directory.

Task

1. Start the RAP databases (RAPCache and RAPStore) by selecting `start` in the RAP OpsConsole.
2. Start the RAP subscribers by selecting `start` in the RAP OpsConsole.
3. From a command prompt, execute the **start.sh** script.

The `start.sh` script executes:

```
esp_rap_out_adapter -f $RAPOUT_HOME/config/espfeedhandler.xml -t  
$RAPOUT_HOME/templates -p $RAPOUT_HOME/config
```

See also

- *Start Command* on page 303

Stopping the RAP Adapter

Once you have configured the adapter, stop it using the **esp_rap_out_adapter** command.

1. Shut down the adapter:
 - If you are running the adapter in the foreground, go to the window in which you started the adapter and press `Ctrl-C`.
 - If you are running the adapter in the background, enter `ps -eaf | grep esp_rap_out_adapter` to get the process ID of the adapter, then enter `kill -ll processID`.
2. Shut down the RAP subscribers by selecting `stop` in the RAP OpsConsole.
3. Shut down the RAP databases by:
 - Selecting `stop` in the RAP OpsConsole.

- If you are running the adapter in the background, enter `ps -eaf | grep dataserver` (for RAPCache) or `ps -eaf | grep IQSRV15` (for RAPStore) to get the process ID of the databases, then enter `kill -ll processID..`

See also

- *Stop Command* on page 304

Replication Server Adapter

The Sybase Replication Server adapter replicates and synchronizes database transactions.

Configuring the Adapter on the Replication Server Workstation

Set up the Replication Server adapter, schema, and source location on the Replication Server workstation.

The adapter copies information from one database to another. These configuration instructions are compatible with source data coming from the Sybase Adaptive Server Enterprise database, as well as other databases.

1. Set up the replication system according to the Sybase Replication Server documentation.
2. Using the **dsedit** utility, add an entry to the interfaces (`sql.ini`) file with the name of the Event Streaming Processor workstation and the port used for the Replication Server adapter connection. This entry used to specify the **Replication Server adapter data server name** and **TDS Port** in the Replication Server Adapter configuration process. See *Chapter 6: Using dsedit* in the *Sybase Adaptive Server Enterprise 15.7 Utility Guide* for more information on modifying the interface or `sql.ini` files. For example, if the adapter and the Event Streaming Processor are on a workstation named `my_workstation` and the connection is to be made on port 5100, use:

```
[RSadapter]
query=TCP,my_workstation,5100
```

3. Define the user name and password.

```
create user rsuser
set password rspassword
go
```

4. Create the connection from the Replication Server to the adapter. Use the same server name used in the previous step (and later for the Replication Server adapter configuration). Log in to the Replication Server to create the Replication Server adapter connection. For example,

```
create connection to RSadapter.RSadapter
set error class to srsa_error_class
set function string class srsa_function_class
set username rsuser
set password rspassword
set batch to "off"
with dsi_suspended
go
alter connection to RSadapter.RSadapter
```

```
set replication server error class to srsa_rs_error_class
go
```

To turn off minimal columns,

```
alter connection to RSadapter.RSadapter
set replicate_minimal_columns to 'off'
go
```

Note: Do not use minimal columns in the **repdef**.

To enable batching,

```
alter connection to RSadapter.RSadapter
  set batch to 'on'
go
alter connection to RSadapter.RSadapter
  set dsi_cmd_separator to ';'
go
```

Define the user name and password used for this connection within the Replication Server. This user name must not be the same as that of the administrator user of the Replication Server adapter.

5. Create the replication definitions. A replication definition specifies the schema and the source location for a given table or stored procedure. Log into the Replication Server to create the sample TEST replication definition.

There is a replication definition for a source table named "TEST" (create table TEST (ID int, FNAME char(15)), which is defined on a sourced database located on an Adaptive Server Enterprise server named ASEHOST.

```
create replication definition TESTrep
with primary at ASEHOST.sourcedb
with all tables named 'TEST'
(ID int, FNAME char(15))
primary key (ID)
go
```

6. Mark the Adaptive Server Enterprise source "TEST" table for replication. Log in to the Adaptive Server Enterprise server, locate the source table "TEST", and execute:

```
esp_setreptable 'TEST', true
go
```

7. Define the subscriptions. Each subscription defines a target for the information coming through the Replication Server. In the following example, the target is the Replication Server adapter connection for the Replication Server adapter located on the Event Streaming Processor workstation. Log in to the Replication Server to create the sample TEST subscription:

```
create subscription TESTsub
for TESTrep
with replicate at RSadapter.RSadapter
without materialization
go
```

Configuring the Adapter on an Event Streaming Processor Workstation

Set up a project using the Replication Server adapter on an Event Streaming Processor workstation.

Prerequisites

Complete the Replication Server adapter configuration on the Replication Server workstation.

Task

1. Start the Studio:

Windows	Click Start > All Programs > Sybase > Event Stream Processor > Studio
UNIX	Enter <code>\$ESP_HOME/ESP/studio/esp_studio</code>

2. Define a new project using the Studio Visual editor. In the Studio, select **File > New > Project**.
3. Configure the Replication Server Input Adapter.
 - a) Drag the Replication Server Input Adapter from the Input Adapters Palette to the Canvas.
 - b) Click the Edit Properties icon.
 - c) From the Configure Adapter Properties window that opens, configure the adapter parameters used to connect to the Replication Server System Database (RSSD) to obtain metadata on tables and stored procedures:

RSSD contains information about the databases involved and the data transfer rules.


Property Label	Property ID	Type	Description
RSSD Host	rssdHost	string	(Required) The RSSD server port.
RSSD Port	rssdPort	uint	(Required) Specify the port to use on the server containing the RSSD database.
RSSD Database Name	rssdDatabase-Name	string	(Required) Specify the name of the RSSD database created by the Replication Server to store replication information.

Property Label	Property ID	Type	Description
RSSD User Name	rssdUser	string	(Required) Specify the user name used to connect to the RSSD server during schema discovery. This user must have permissions to run the RSSD Stored Procedures.
ESP Server User ID	espUser	string	(Required) User name for connecting to the ESP Server.
RSadapter Data Server Name	repSubscription-Server	string	(Required) Specify the shared data server and database name that defines the replication server connection pointing to the Replication Server adapter within Event Streaming Processor. This server name is also used to define an entry in the Replication Server's interfaces <code>sql.ini</code> file. This value should be used for both the "data server" and "database name" portions of the replication server connection definition.
Project URI	projectUri	string	(Required) URI to connect to a project in a cluster environment.
Authentication Mechanism Type	authType	string	(Required) Specify the authentication mechanism to use.
ESP Server Password	espPassword	string	(Optional) ESP Server password for the ESP Server User ID. Do not use if using RSA authentication.
RSA Key Store	rsaKeyStore	string	(Optional) RSA Key Store file name and location.
RSSD Password	rssdPasswd	password	(Optional) Specify the password for the RSSD user.
RSA Key Store Password	rsaKeyStorePassword	password	(Optional) RSA Key Store password.

4. Select the **Advanced** tab and configure the adapter parameters for runtime processing and internal communications:

Property Type	Property ID	Type	Description
Stored Proc Stream Operation	storedProcStreamOp	choice	(Advanced) Specify whether an insert or upsert operation is performed with stored procedure for the stream in Replication Server. Defaults to insert.
TDS Port	tdsListenerPort	uint	(Advanced) Specify the port used by the adapter. This is the port to which the Replication Server connection definition must connect. This port is defined within the interfaces (<code>sql.ini</code>) file on the Replication Server workstation it defines connectivity between the Replication Server and the adapter.
Adapter Admin User	adminUser	string	(Advanced) Specify a user name for the adapter to use for internal communications. This name can be anything except it must not match the user defined within the Replication Server connection definition.
Adapter Admin Password	adminPasswd	password	(Advanced) Specify a password associated with the user name the adapter uses for internal communications.
Transactional Stream Operations	isTransactional	boolean	(Advanced) Specify whether or not Event Streaming Processor data changes are in a Sybase transaction. This causes changes to be committed to a log store immediately after the transaction. This parameter should be set to true when using a log store for a persistent rs_lastcommit .
Async Stream Operations	isAsync	boolean	(Advanced) Specify whether or not Event Streaming Processor data changes are in an Sybase transaction. This parameter allows you to use asynchronous stream operations.
Batched Stream Operations	isBatched	boolean	(Advanced) Specify whether or not the Replication Server adapter sends data to the Sybase Event Streaming Processor in batches.
Batch Size	batchSize	uint	(Advanced) If the Batched Stream Operations parameter is set to true, specify the number of rows in the batch.

Property Type	Property ID	Type	Description
Publish When Batch is Full	publishWhenBatchFull	boolean	(Advanced) If set to true, the adapter writes data to the stream when the batch reaches Batch size . If set to false, the adapter waits for a commit to write data to the stream.
Error on Missing Stream Column	errorOnMissingStreamColumn	boolean	(Advanced) Set to true to send an error back to the Replication Server if a column in the repdef is not defined within the stream. Sybase recommends you set this parameter to false (to avoid frequent disconnections from the replication server) is recommended.
RSSD Table Name	tableName	tables	(Advanced) Specify the name of the table in RSSD. Required if the RSSD table name is different from the corresponding stream name.
Field Mapping	permutation	permutation	(Advanced) Specify mapping between the internal and external fields.
Property Set	propertyset	string	(Advanced) (Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.

5. Select **OK** to save the project configuration.
6. Discover the source tables or stored procedures for the adapter. The schema discovery process establishes a connection to the RSSD and reveals defined subscriptions that target the Replication Server adapter.
 - a) Click **Schema Discovery**  in the Adapter shape.
A list of tables or stored procedures that have Replication Server adapter subscriptions associated with them is returned. The `rs_lastcommit` table is also returned.
 - b) Select one of the discovered tables and click **Next**.
 - c) In the Create Element dialog, choose **Create new input window**.
 - d) Click **Finish**.
 - e) Define primary keys. Each table and stored procedure stream must contain at least one primary key. Sybase recommends that this key match the primary key defined in the

replication definition. Locate the appropriate primary key column and click on the toggle key to the left of it. The toggle key changes to an image of a key.

Schema discovery adds a special column named 'ra_pkey' to the stream definition for a stored procedure. The 'ra_pkey' column must be set as the primary key and 'Autogen' set to true for stored procedures.

- f) Select **File > Save** to save the changes.
7. Test data movement to the Event Stream Processor stream. Use the Studio for these steps, unless otherwise noted:
 - a) Open the project within the Studio.
 - b) Go to the Run-Test perspective.
 - c) Run this project in a pre-started cluster. The cluster URI, workspace name, and project name must match the projectURI which is defined in adapter parameters. A successful startup appears as:
Stream TEST is ready for Replication Server connections.
 - d) Log in to the Replication Server and resume the Replication Server adapter connection:

```
resume connection to RSadapter.RSadapter
go
```
 - e) Insert sample data into the source table.
 - f) Verify that the replicated data reaches the Replication Server adapter stream using the Stream View tab in Studio.

Defining a Persistent rs_lastcommit

The rs_lastcommit table is non-persistent by default. It is held in memory and cleared when the stream is shut down. This results in a full replay of all remaining items within the Replication Server when the stream is restarted. Sybase recommends making rs_lastcommit persistent to minimize the replay of transactions following a stream restart.

In Studio:

1. Add rs_lastcommit to the project.
 - a) Select the rs_lastcommit table returned as part of the schema discovery process and click **Next**.
 - b) Choose **Create a new input window** and click **Finish**.
 - c) Set the "origin" column as the primary key.
2. Configure rs_lastcommit to use the persistent log store.
 - a) In the **Palette**, expand **Shared Components**.
 - b) Select and drag the **Log Store** component from **Shared Components** over to the project.
 - c) Edit the store property in rs_lastcommit so that it selects the log store.

Supported Datatypes

Map Adaptive Server Enterprise and Replication Server datatypes to Event Stream Processing datatypes.

ESP Datatype	Adaptive Server Enterprise and Replication Server Datatypes
integer	smallint, tinyint, int, bit
timestamp	datetime, time
date	date, smalldatetime
long	bigint, unsigned bigint, unsigned int, unsigned smallint
string	binary, char, unichar, nchar, nvarchar, varbinary, univarchar, varchar, timestamp
float	numeric, float, real
money	money, smallmoney
bigdatetime	bigdatetime (ASE15.7 or later)
boolean	bit
binary	varbinary

Performance Tips

Modify the system configuration to improve performance.

- Configure the Replication Server to use batches (terminator must be a semicolon).
- Configure the Replication Server to use larger packets. For example, set a value of 4096.
- Run on a 64-bit machine.
- Set all log levels to info or lower in the `$ESP_HOME/adapters/repserver/config/log4j.properties` file.

Reuters Marketfeed Adapter

The Sybase Reuters Marketfeed adapter is a software interface between Event Stream Processor and the Reuters Market Data System (RMDS). It uses the Reuters Marketfeed message format.

You can configure the adapter as an input or output adapter. The input adapter subscribes to one or more Reuters Instrument Codes (RICs) on the RMDS to provide input to Event Stream Processor. The output adapter publishes output from Event Stream Processor to the RMDS.

CHAPTER 2: Adapters Supported by Event Stream Processor

This enables Event Stream Processor to use the speed and reliability of Reuters' infrastructure to deliver data.

The Reuters Marketfeed Input adapter supports schema discovery. Run two adapter instances if you require both input and output capabilities.

The adapter runs on Solaris and Linux operating systems but you can use it with Event Stream Processor software running on Solaris, Linux, or Windows.

Requirements

The Reuters Marketfeed input and output adapters have several requirements.

An input adapter requires:

- A RMDS market data connection that uses the Marketfeed protocol
- A working subscription for data on one or more financial instruments

An output adapter requires:

- A working connection with support for sending data to RMDS using the Marketfeed protocol

General Configuration

Enable user access for each user account that runs the Reuters Marketfeed adapter, and configure an input connection from Reuters and an output connection to Reuters.

Enabling User Access

Enable user access for each user account that uses the Reuters Marketfeed adapter.

1. Ensure the user account has permission to execute the installed software.
2. Add an environment variable, `$ESP_REUTERS_HOME`, set to the root of the adapter hierarchy, to the user's runtime environment.
3. (Optional) Add the environment variable to your shell profile.
4. Event Stream Processor supports RSA, LDAP, and Kerberos authentication. If your installation uses one of these authentication methods, ensure the user account is set up to work with that method of authentication.

Configuring an Input Connection from Reuters

Modify the sample configuration file for your site's RMDS connection.

Prerequisites

- Create (or choose) a directory in which to store your site-specific configuration files.
- Create an environment variable (`MY_CONFIG`) and set it to the full path name of that directory.

Task

During the installation process, a sample configuration file (`rfa7sub.cfg`) was placed in the `$ESP_REUTERS_HOME/config` directory. This file, shown below, follows the Reuters format for configuration files.

```
# Change this if necessary.
# the port number of the P2PS (default 8101)
\Connections\Connection_SSLED\PortNumber           = 8101

# Change this if necessary.
# the user name to connect with (should be the DACS name if DACS is
enabled)
\Connections\Connection_SSLED\UserName             = "triarch"

# Change this if necessary.
# a list of P2PS host names
\Connections\Connection_SSLED\ServerList           = "localhost"

# Refer to RFA documentation for more advanced changes to the
remaining entries
\Connections\Connection_SSLED\connectionType       = "SSLED"

\Adapters\SASS3_Adapter\requestQueueReadThreshold = 1
\Adapters\SASS3_Adapter\mainLoopTimerInterval     = 200

\Adapters\SSLED_Adapter\masterFidFile              = "config/
appendix_a"
\Adapters\SSLED_Adapter\enumTypeFile              = "config/
enumtype.def"
\Adapters\SSLED_Adapter\downloadDataDict          = false

# Change the fileLoggerFilename appropriately for your setup
\Logger\AppLogger\windowsLoggerEnabled            = false
\Logger\AppLogger\fileLoggerEnabled               = true
\Logger\AppLogger\fileLoggerFilename              = "rfa7sub.{p}.log"

\Control\Entitlements\dacs_SbeEnabled               = false
\Control\Entitlements\dacs_CbeEnabled              = false

\Logger\ComponentLoggers\Connections\messageFile  = "config/
messages/RFA7_Connections.mc"
\Logger\ComponentLoggers\Adapter\messageFile      = "config/
messages/RFA7_Adapter.mc"
\Logger\ComponentLoggers\SessionCore\messageFile  = "config/
messages/RFA7_SessionLayer.mc"
\Logger\ComponentLoggers\SSLED_Adapter\messageFile = "config/
messages/RFA7_SSLED_Adapter.mc"

\Sessions\Session1\connectionList                 =
"Connection_SSLED"
```

1. Obtain this information from your system administrator:

CHAPTER 2: Adapters Supported by Event Stream Processor

- Name of the server that receives Marketfeed data from RMDS
 - Port number on the machine to which your system connects
 - User name defined for your connection to Reuters
 - Name of each Reuters service to which you are subscribed
2. Make a copy of the sample configuration file in your `$MY_CONFIG` directory:

```
cp $ESP_REUTERS_HOME/config/rfasub.cfg $MY_CONFIG
```
 3. Use a text editor to open the configuration.
 4. In the `\Connections\Connection_SSLED\PortNumber` line, replace the default port number (8101) with the number used by your Reuters connection, if different.
 5. In the `\Connections\Connection_SSLED\UserName` line, replace `triarch` with the user name for your Reuters subscription. Keep the surrounding quotation marks. In the `\Connections\Connection_SSLED\ServerList` line, replace `localhost` with the name of the server that receives Marketfeed data from RMDS. Keep the surrounding quotation marks.
If your system has more than one server receiving data from RMDS, include all of their names in a comma-separated list, in priority order.
 6. (Optional) In the `\Logger\AppLogger\fileLoggerFilename` line, you can change the name of the log file.
The default file name specified here, `rfasub.{p}.log`, includes the string `{p}` which the Reuters library replaces with the UNIX process ID when it creates the log file.
 7. Save the modified file.
The other parameters in the configuration file also affect the functioning of the Reuters Marketfeed adapter, and you may want to modify them as well.

Configuring an Output Connection to Reuters

Modify the sample configuration file for your site's RMDS connection.

Prerequisites

- Create (or choose) a directory in which to store your site-specific configuration files.
- Create an environment variable (`MY_CONFIG`) and set it to the full path name of that directory.

Task

During the installation process, a sample configuration file (`rfaPub.cfg`) was placed in the `$ESP_REUTERS_HOME/config` directory. This file, shown below, follows the Reuters format for configuration files.

```
# Change this if necessary.
# This needs to match port number for the route as defined in the
Source Distributor.
\Connections\Connection_SSLED_MP\ipcServerName      = "8105"
```

```

# Refer to RFA documentation for more advanced changes to the
remaining entries.
\Connections\Connection_SSLED_MP\connectionType      = "SSLED_MP"
\Connections\Connection_SSLED_MP\entitlementData      = false
\Sessions\Session1\connectionList                    =
"Connection_SSLED_MP"

# Change the fileLoggerFilename appropriately for your setup
\Logger\AppLogger\windowsLoggerEnabled               = false
\Logger\AppLogger\fileLoggerEnabled                  = true
\Logger\AppLogger\fileLoggerFilename                  = ".\rfapub.
{p}.log"

\Control\Entitlements\dacs_SbeEnabled                  = false
\Control\Entitlements\dacs_CbeEnabled                  = false

\Logger\ComponentLoggers\Connections\messageFile     = "./config/
messages/RFA7_Connections.mc"
\Logger\ComponentLoggers\Adapter\messageFile         = "./config/
messages/RFA7_Adapter.mc"
\Logger\ComponentLoggers\SessionCore\messageFile    = "./config/
messages/RFA7_SessionLayer.mc"
\Logger\ComponentLoggers\SSLED_Adapter\messageFile   = "./config/
messages/RFA7_SSLED_Adapter.mc"
\Logger\ComponentLoggers\SSLED_MP_Adapter\messageFile = "./config/
messages/RFA7_SSLED_MP_Adapter.mc"

```

1. Obtain this information from your system administrator:
 - Port number at which the src_dist or RMDS infrastructure server listens for updates from the Reuters Marketfeed adapter
 - Name of the server that receives updates from Event Stream Processor
2. Make a copy of the sample configuration file in your \$MY_CONFIG directory.

```
cp $ESP_REUTERS_HOME/config/rfapub.cfg $MY_CONFIG
```
3. Use a text editor to open the configuration.
4. In the \Connections\Connection_SSLED_MP\ipcServerName line, replace the default port number (8105) with the port number at which your src_dist listens for updates from the Reuters Marketfeed adapter, if different.
5. (Optional) In the \Logger\AppLogger\fileLoggerFilename line, change the name of the log file. The default file name specified here, .\rfapub.{p}.log, includes the string {p} which the Reuters library replaces with the UNIX process ID when it creates the log file.
6. Save the modified file.

Input Adapter Configuration

Configure an input adapter to push data from the Reuters Market Data Service (RMDS) to Event Stream Processor.

Before configuring an input adapter, decide what data you need and how you want to set up your system.

CHAPTER 2: Adapters Supported by Event Stream Processor

You need to know the following about the Event Stream Processor instance from which you receive data.

- Possible security options in a cluster environment, and the workspace and project name.
- What type of authentication mechanism (Kerberos, RSA, LDAP, or none) does it use?

Data Decisions

Decide how the incoming Reuters data fits into the project.

Also decide whether you require Level 1 or Level 2 data. For Level 1 data, use the Reuters Marketfeed adapter, and for Level 2 data, use the Reuters OMM adapter instead.

Decision	Description
Venues	Decide which venues are of interest (for example, NYSE, NAS-DAQ, Toronto, and so on).
RICs and FIDs	Determine what market data you need. Specifically, which Reuters Instrument Codes (RICs) you want the adapter to provide to Event Stream Processor, and which Reuters Field IDs (FIDs) for these instruments you want to use.
Streams	The Reuters adapter can furnish data to one or more streams on Event Stream Processor. To use the Reuters Market Data provided by the adapter, decide which existing data streams to map to the adapter's data feed, or define one or more new streams.

Administrative Decisions

You have several administrative decisions to make in regards to the project.

Decision	Description
Session Name	An arbitrary string used to link the project and the adapter map file. Use it consistently.
Directories for logging and stream output	The adapter writes its own log messages and can generate a separate set of Reuters log messages. In the configuration, specify if and where these log files should be written.
Sybase user account	Specify a valid Event Stream Processor user account for the adapter to use, unless you specified no authentication when you started the Event Stream Processor.

Input Adapter Map File

The map file configures the interface between the Reuters Marketfeed adapter and Event Stream Processor. It specifies which source streams receive data from RMDS via the adapter, and it maps specific RMDS Field Identifiers (FIDs) to specific columns in that source stream.

The input adapter map file must accomplish three major tasks:

- Match incoming data elements to columns in one or more streams defined in the Event Stream Processor configuration file.
- Match the RIC provided with each update from the adapter with a row in the Event Stream Processor configuration file.
- Ensure that each update from the adapter can be converted into a record that provides a unique key for each stream being populated, as defined by the stream's column definitions.

Data Structures

Data structures have three important structural aspects: data columns, datatypes, and key values.

- Each data stream includes one or more data columns.
- Each column has a datatype.
- In most streams, each row has a unique key value. The source stream definition designates one or more columns as "key" columns.

Incoming RMDS Data

When the adapter subscribes to RMDS for a certain RIC, RMDS first sends an initial image containing all available market data for that RIC.

After that, RMDS sends an update when any values for a subscribed RIC change. Each update consists of the identifying RIC, with the Field Identifier (FID), and the new value for each change. Each FID defined for RMDS has a datatype.

Market Data Field Mapping

Map each column in the target Event Stream Processor stream to a Reuters FID or a pseudo-field.

Find the appropriate FID for each column in the stream. The datatype of the Event Stream Processor column must be compatible with the datatype of the Reuters FID that feeds it.

Here are possible matches between FID datatypes and Event Stream Processor datatypes:

Event Stream Processor Datatype	Reuters Datatype
integer or long	integer
string	alphanumeric

Event Stream Processor Datatype	Reuters Datatype
integer	enumerated
timestamp or date	time, date
money or float	price
long or integer	time_seconds
Not supported	binary

Reuters Instrument Code Mapping

The identifier of each incoming RMDS update is the Reuters Instrument Code (RIC).

Map the RIC to a column of datatype string in the stream. If the stream you want to map to does not have a suitable column, either add a column to the stream, or map to a different stream.

Matching the Stream's Key

The adapter map file must configure the adapter so that every update sent to the Event Stream Processor stream includes a field or combination of fields conforming to the unique key defined for that stream. To make this more flexible, the adapter configuration mechanism supports "pseudofields".

The market data updates that the adapter receives from RMDS are mapped to columns in the Event Stream Processor stream using the dataField or dateTimeField element in the map file. RMDS also provides non-market data information, for example, each update includes a RIC. Additionally, you can configure the adapter to add a sequence number to each update.

To make these data items available to the mapping process, the map file mechanism supports the following elements, called pseudofields:

Field	Description	Datatype
itemName	The RIC.	string (required)
serviceName	Name of the service from which RMDS received the market data from this RIC.	string (optional)
itemState	The item state.	integer (optional)
sequenceNumber	A unique number, assigned sequentially by the adapter to each incoming event (whether or not it causes an update).	long (optional)
FIDListField	Shows the FID name and value for each updated value.	string (optional)

Field	Description	Datatype
updateNumber	A unique number, assigned sequentially by the adapter to each incoming update.	long (optional)

Getting Stream Information from the Project

Gather the necessary information about the Reuters stream.

The first step in configuring the input adapter is to determine the source streams on Event Stream Processor which will receive the RMDS market data. If the Event Stream Processor project does not already include one or more streams for this purpose, define a new stream (or streams) for use with the Reuters adapter.

After you have chosen (or defined) the streams that will receive data from the Reuters Marketfeed adapter, collect information about that stream from your project file. The Event Stream Processor project file contains one or more stream definitions. Each stream definition specifies a data stream that is instantiated when Event Stream Processor is started. The stream definition comprises:

- A unique ID for the stream
- A database store and output file for the stream data
- A list of the columns used as the unique key value for each row in the data stream

Once you have decided which streams will carry the RMDS data provided by the Reuters adapter, get information from the stream definition in the project file. There is no standard for project file names. Two Event Stream Processor installations may have completely different stream definitions, but the definition of any stream includes the same basic set of components.

These instructions refer to the example project to show what components of the stream configuration you must identify to configure the Reuters Marketfeed adapter.

1. Open the project to which the adapter provides data. The example shown here is the `$ESP_REUTERS_HOME/examples/example.ccl` file supplied with the Reuters Marketfeed adapter distribution.
2. Find the name of the source stream. The opening `SourceStream` tag specifies the name of the stream as the value of the `id` attribute. The first source stream in this example is named “stream1.”

The stream used for subscription by the Reuters Marketfeed adapter must always be a source stream.

3. Determine the key fields. Examine each of the column entries between the opening and closing `SourceStream` tags to see if the `key` attribute is set to true. In this example, “stream1”, has one key field, “symbol.”
4. Carefully note the number and order of the column entries in the source stream definition. In the input adapter map file, list the same set of data in the same order.

Creating the Input Adapter Map File

Create an adapter map file to configure the interface between the Reuters Marketfeed input adapter and Event Stream Processor.

This procedure maps updates from RMDS to the source stream defined in the `example.ccl` file. This file is in the `$ESP_REUTERS_HOME/examples` directory along with an example map file.

1. Open a new map file using an editor.
2. Enter the following as the first line of the file to specify that the adapter map file conforms to XML version 1.0.:

```
xml version="1.0" encoding="UTF-8"
```

3. Specify that this is an adapter map file and that includes a separate file:

```
<!DOCTYPE adapter [
<!ENTITY rmdsFields SYSTEM "rmds.sm.mf.xml">
]>
```

4. Add the opening and closing adapter tags. In the opening adapter tag, specify the name of the adapter. For example:

```
<adapter name="mySubscribeAdapter1">
</adapter>
```

5. After the opening adapter tag, add the publication element. Specify the name to be used in log messages for this adapter and any other attributes required to prescribe how the adapter should deliver data to Event Stream Processor.

For example,

```
<publication name="RMDS Adapter - low latency" retryInterval="5" />
```

This example also includes a `retryInterval` attribute with a value that tells the adapter to wait five seconds before retrying if it fails to connect to Event Stream Processor.

6. After the publication element, add the opening and closing `streamMaps` tags to contain the `streamMap` elements that do the actual mapping between RMDS FIDs and columns of an Event Stream Processor stream. Each `streamMap` maps to one and only one Event Stream Processor stream.

```
<streamMaps>
</streamMaps>
```

Since the `streamMaps` section can contain more than one `streamMap`, one instance of the adapter can provide RMDS data to more than one Event Stream Processor stream.

7. Enter a `streamMap` element for each Event Stream Processor stream to which you wish to send RMDS data. For each `streamMap`,
 - a) Enter the opening `streamMap` tag specifying the name of the Event Stream Processor stream to which the RMDS data is sent as the value of the `name` attribute.
 - b) Enter the closing `streamMap` tag.

- c) Between the streamMap tags, add one mapping element for each column defined in the target stream's definition. You can do this in the map file itself or in a separate file that is included in the map file as an entity.

```
<streamMap name="stream1" flags="NO_SHINE">
&rmdsFields;
</streamMap>
```

8. After the streamMaps section, add the rfa element, including:

- A config attribute that specifies the absolute path and file name of the Reuters configuration file
- A sessionName attribute that specifies a session name corresponding to the one used in the Reuters configuration file

```
<rfa config="$ESP_REUTERS_HOME/config/rfasub.cfg"
sessionName="Session1" />
```

The rfa element may also include attributes to modify the adapter's treatment of blanks (by default it converts them to zeros). You can specify the value for the blank attribute or specify values for each datatype directly using the blankInt32, blankInt64, blankMoney, blankString, blankDate, and blankTimestamp attributes. Specify a value that does not conflict with any of the values you expect in your data. If you are using both input and output adapters, specify the same value for each attribute to both adapters.

9. Between the rfa element and the closing adapter tag, add the opening and closing itemLists tags. When entering the opening itemLists tag:

- Specify the Reuters service from which the adapter is receiving RMDS data as the value of the service attribute.
- Specify the name of the Event Stream Processor stream that is receiving the RMDS data as the value of the stream attribute.

```
<itemLists service="IDN_RDF" stream="stream1">
</itemLists>
```

The itemLists tags will contain one or more pairs of opening and closing itemList tags.

10. Between the itemLists tags, add opening and closing itemList tags for each separate list of RICs to which the adapter subscribes.

11. Between the itemList tags, add an item element for each RIC to add to the list. When entering the item element:

- Specify an RIC to which the adapter subscribes as the value of the name attribute.
- (Optional) Specify the name of the queue you wish to use as the value of the rfaQueue attribute. Specifying an rfaQueue spawns a separate thread to do the processing.
- (Optional) Specify the name of the service to use.

For example:

```
<itemList>
<item name="AAPL.O" rfaQueue="queue1" />
<item name="CSCO.O" />
</itemList>
```

Running the Input Adapter

Run the Reuters Marketfeed input adapter once you have configured it.

Prerequisites

Configure an adapter.

Task

1. Ensure that **esp_server** is running and that the project has been loaded and started.
2. Start the adapter using the **esp_rmds** command.
 - a) If the Event Stream Processor is running with no authentication, start the adapter with this command:

```
esp_rmds -a in -f mapfile -p cluster_host:cluster_port/  
workspace/project
```

using the appropriate mapfile, cluster_host, cluster_port, workspace, and project names for the project to which the adapter will connect.

- b) If the Event Stream Processor is running with some form of authentication, refer to *Command Usage* to obtain the additional arguments necessary for the command to start the adapter.

The exact usage of the command depends on how you started your Event Stream Processor. You must invoke the adapter with compatible options. The command string shown above invokes neither encryption nor authentication: you can specify either or both.

3. The adapter starts the subscription by first connecting to Event Stream Processor and then connecting to RMDS. Both connections must be operational for any data to flow.

If you plan to direct the adapter's log output to stderr, as shown here, you may want to redirect stdout and stderr to a log file (for example, append `>& myrmdslog &` to the command line shown above).

Testing the Adapter

If the adapter is not working as expected, you can perform a quick sanity check by executing the **esp_rmds** command and verifying whether the adapter is sending Reuters market data to Event Stream Processor.

- Execute **esp_rmds**:

```
esp_rmds -v
```

This command returns the version information. Ensure that the Event Stream Processor to which you are connecting is compatible with your version of the adapter.

- There are three quick ways verify that the Reuters Marketfeed adapter is sending Reuters Market Data to Event Stream Processor:

- Use the Studio or the **esp_subscribe** command to check the output of the stream configured to receive Reuters data.
- Use the **tail** command on the redirected adapter log file (specified in the adapter map file) or the Reuters subscriber log (specified in the configuration file `rfasub.cfg`) for activity.
- Run the **esp_rmds** command with the `-d7` option to produce verbose output.

Multiple RICs

When configuring an input adapter, you will usually want to specify multiple RICs.

There are several ways to do this:

- Specify each individual RIC by entering the name directly into the map file or use an XML ENTITY include file.
- Specify a chain RIC from Reuters.
- Create a dynamic watch list, which employs Event Stream Processor to specify the list of RICs.
- Use a combination of the options above.

Individual RICs

Enter an item element declaration for each RIC you want in the `itemList` section of the map file.

Here is an example of this:

```
<itemLists service="SSL_PUB" stream="stream1">
<itemList>
<item name="CSCO.O" />
<item name="K.N" />
<item name="KBN.N" />
<item name="KBR.N" />
<item name="ACAM.ARC" />
<item name="IBM.ARC" />
</itemList>
</itemLists>
```

It can become difficult to create and maintain your list of RICs this way if it is very large or changes frequently, for example, if you are attempting to track all of the stocks traded on the NYSE. All RICs for the same stream must use the same FID set. Since FIDs often vary by venue, use a different `itemList` and `streamMap` for each venue.

Chain RIC

When you specify the name of a chain RIC, Reuters translates it to a list of individual RICs. Chain RICs usually contain all of the RICs from a single market or for a single index instrument, such as the S&P 500 or the Russell 2000.

For example, to specify the chain RICs for the Dow Jones Index and the SIAC entities, add a `chain-map` section:

```
<streamMap name="chainMap" chain="1" >
<itemName /> <dataField name="REF_COUNT" />
```

```

<dataField name="NEXT_LR" /> <dataField name="PREF_LINK" />
<dataField name="LINK_1" /> <dataField name="LINK_2" />
<dataField name="LINK_3" /> <dataField name="LINK_4" />
<dataField name="LINK_5" /> <dataField name="LINK_6" />
<dataField name="LINK_7" /> <dataField name="LINK_8" />
<dataField name="LINK_9" /> <dataField name="LINK_10" />
<dataField name="LINK_11" /> <dataField name="LINK_12" />
<dataField name="LINK_13" /> <dataField name="LINK_14" />
<dataField name="LONGNEXTLR" /> <dataField name="LONGPREVLR" />
<dataField name="LONGLINK1" /> <dataField name="LONGLINK2" />
<dataField name="LONGLINK3" /> <dataField name="LONGLINK4" />
<dataField name="LONGLINK5" /> <dataField name="LONGLINK6" />
<dataField name="LONGLINK7" /> <dataField name="LONGLINK8" />
<dataField name="LONGLINK9" /> <dataField name="LONGLINK10" />
<dataField name="LONGLINK11" /> <dataField name="LONGLINK12" />
<dataField name="LONGLINK13" /> <dataField name="LONGLINK14" />
</streamMap>

```

and enter their names in the `itemList` section.

```

<itemList stream="stream1" service="IDN_RDF" >
<item name="0#.DJI" /> <!-- The Dow Jones Index -->
<item name="0#SIAC" /> <!-- The entities of SIAC -->
</itemList>

```

For more details about chains, look at the example in `chain.example.map.xml` in the `$ESP_REUTERS_HOME/examples` directory. For more information about Reuters chain RICs, see the *Reuters Venue Guide* for your chosen venue, which is available from Reuters.

Creating a Dynamic Watch List

Creating a dynamic watch list is a bit more complex than creating individual or chain RICs, but is also more flexible. Chain RICs are limited to those defined by Reuters, but with this method you can specify a customized list of RICs.

Prerequisites

Define a source stream (named `MyInfoStream`) to receive the data, and manually edit the list of RICs to include.

Task

Creating a dynamic watch list is also dynamic: when inserts or deletes occur on the stream configured using these steps, RMDS subscriptions to the appropriate RICs are started or stopped.

1. Define a stream on Event Stream Processor (for example, `MyListStream`), which publishes to the adapter the list of RICs to which you want to subscribe. This stream requires these columns:

Column	Description
symbol	Specifies an RIC symbol ticker (for example, CSCO.O) to which the adapter should subscribe.
service	Specifies the RMDS service on which to subscribe to obtain data for that RIC.
stream	Specifies the name of the stream (for example, MyInfoStream) on which the adapter publishes data for this RIC.

The stream can also include an optional fourth column, rfaQueue.

2. Define a second stream on Event Stream Processor (for example, MyInfoStream) that receives the data requested by the first stream.
3. Edit the map file to include the subscription.

```
<subscriptions>
<subscription name="subscription1" flags="BASE" >
<stream name="MyListStream" >
<name column="3" /> <!-- symbol -->
<field column="1" name="service"/>
<field column="2" name="stream"/>
</stream>
</subscription>
</subscriptions>
```

4. Specify the set of RICs you want and send them to the first stream you created (for example, MyListStream) to subscribe to them.

- a) Create a file with the same six columns that the stream expects in comma-separated values (CSV) format. The columns are: stream from which you are receiving data, opcode, service, symbol, and stream to which you are sending data.

For example, open a new file (RIClist.csv) using an editor and put in these lines.

```
MyListStream,p,,IDN_RDF,MyInfoStream,CSCO.O
MyListStream,p,,IDN_RDF,MyInfoStream,K.N
MyListStream,p,,IDN_RDF,MyInfoStream,KBN.N
MyListStream,p,,IDN_RDF,MyInfoStream,KBN.R
MyListStream,p,,IDN_RDF,MyInfoStream,ACAM.ARC
MyListStream,p,,IDN_RDF,MyInfoStream,IBM.ARC
```

- b) Send the data from the file to Event Stream Processor using the **esp_convert** and **esp_upload** commands. The following example assumes that you have installed all Sybase command line tools in the default directories and added those directories to your PATH variable. If you have not, prepend the appropriate path to each command shown in this example.

For example, to send the file created in the previous step to Event Stream Processor running a project named p1 in workspace ws1 on port 19011 of your local server, enter:

CHAPTER 2: Adapters Supported by Event Stream Processor

```
cat RIClist.csv | esp_convert -c user:password -d "," -p  
localhost:19011/wsl/pl | esp_upload -c user:password -p  
localhost:19011/wsl/pl
```

c) Start the adapter:

```
esp_rmids -f mapfile -d7 -c user:password -p localhost:19011/  
wsl/pl >& logfile &
```

If the adapter and Event Stream Processor are on different machines, enter the name of the remote host in place of localhost after the -p in the previous command.

Output Adapter Configuration

Configure an output adapter to push data from Event Stream Processor to RMDS, using RMDS as a message infrastructure.

Before configuring an output adapter, decide which data to provide and how you want to set up your system.

You need to know the following about the Event Stream Processor instance from which you receive data.

- Possible security options in a cluster environment, and the workspace and project name.
- What type of authentication mechanism (Kerberos, RSA, LDAP, or none) does it use?

Data Decisions

Identify which columns from which streams in Event Stream Processor to publish data from.

The Reuters Marketfeed adapter can rearrange the columns from a stream in any order. Its output can also include constants, and the published output can include values from more than one stream.

Consider these items when planning the output of the Reuters Marketfeed Output adapter:

- For each stream for which to publish data, you must specify a unique key in the output adapter map file. Since this adapter sends data to RMDS, the unique identifier should be an RIC.
- Each data column you want to publish from any stream must map to a unique FID.
- Data from one column can be repeated in the published output, giving you a way to publish a DateTime value as separate Date and Time values.
- If the stream you are working with receives data about the same FID from more than one service, you can configure the adapter to differentiate these data items by service and transmit each service's data separately.
- The first time the Reuters Marketfeed adapter publishes to RMDS, it publishes values for all the columns for which it is configured. After that initial image, the adapter only publishes updates for individual columns as these updates occur.

Administrative Decisions

You have several administrative decisions to make in regards to the project.

Decision	Description
Session Name	An arbitrary string used to link the project and the adapter map file. Use it consistently.
Directories for logging and stream output	The adapter writes its own log messages and can generate a separate set of Reuters log messages. In the configuration, specify if and where these log files should be written.
Sybase user account	Specify a valid Event Stream Processor user account for the adapter to use, unless you specified no authentication when you started the Event Stream Processor.

Reuters Information

You need several pieces of information from Reuters to enable the Reuters Marketfeed adapter to publish to the RMDS.

- The name of the Reuters service on which the adapter transmits data
- Up-to-date lists of valid Reuters Instrument Codes (RICs) and Field Identifiers (FID) used by RMDS
- The Product Permission Code assigned by Reuters

The adapter does not work with the Reuters Data Access Control System (DACs), so the Product Permission Code is needed to allow access to the information you are transmitting on the RMDS.

A list of FIDs, `$ESP_REUTERS_HOME/config/appendix_a`, has been supplied as part of the Reuters adapter distribution. You can obtain the latest list and other information from your Reuters technical contact.

The datatype of the Event Stream Processor column must be compatible with the Reuters FID datatype that feeds it. This table shows possible matches between Event Stream Processor and FID datatypes:

Event Stream Processor Datatype	Reuters Datatype
integer, long	integer or price
money, float	price
string	alphanumeric
date, timestamp	date or time

Getting Stream Information from the Project

Gather the necessary information from the project.

The first step in configuring the output adapter is determining which data elements from which streams on the Event Stream Processor are to be published. After you have chosen (or defined) a project containing the items for publication over RMDS via the Reuters adapter, collect information from the streams to obtain the data to send to RMDS.

Each stream definition specifies a data stream that is instantiated when Event Stream Processor is started. The stream definition:

- Specifies a unique ID for the stream
- Identifies the columns used as the unique key value for each row in the data stream

Once you have decided which streams will provide the information to be sent to RMDS by the Reuters adapter, get information from the stream definition in the project file. There is no standard for project file names. Two Event Stream Processor installations may have completely different stream definitions, but the definition of any stream includes the same basic set of components.

1. Open the project to which the adapter provides data. The Reuters Marketfeed adapter distribution includes an example project, `$ESP_REUTERS_HOME/examples/example.ccl`.
2. From the definition of each stream defined in the project:
 - a) Obtain the name of the stream from the id attribute in the opening tag of that stream.
 - b) Verify that the key attribute is set to true for the column containing the RIC and note the column. In this example, both “stream1” and “orderbookStream” have the RIC in the column named “symbol,” which is identified as a key field.
 - c) Decide what data, if any, you want the adapter to send to RMDS.
3. Carefully note which streams contain data to send to RMDS, and where in the stream definition it is located.

In the output adapter map file, reference each of the columns you want to publish.

Creating the Output Adapter Map File

Create an adapter map file to configure the interface between the output adapter and Event Stream Processor.

The examples shown below map updates from RMDS to the source stream defined in the `example.project.xml` file.

1. Open a new map file using an editor.
2. Enter the following as the first line of the file to specify that the adapter map file conforms to XML version 1.0.:

```
xml version="1.0" encoding="UTF-8"
```

3. Add the opening and closing adapter tags by entering this as the first line:.

```
<adapter>
</adapter>
```

4. Define the configuration of the adapter's interface to RMDS by adding the rfa tag, with these attributes:

Attribute	Description
config	Specify the full path name of the Reuters configuration file.
fidFile	Specify the full path name of the Reuters-supplied file that lists all of the valid FIDs.
enumFile	Specify the full path name of the Reuters-supplied file that lists each enumerated type along with the range of values it can take.
serviceName	Specify the service name provided by Reuters for the adapter to send data to RMDS.
sessionName	Specify the sessionName value found in the Reuters configuration file, rfasub.cfg.

For example, using the files that were shipped with the adapter distribution:

```
<rfa config="$ESP_REUTERS_HOME/config/rfapub.cfg"
fidFile="$ESP_REUTERS_HOME/config/appendix_a"
enumFile="$ESP_REUTERS_HOME/config/enumtype.def"
serviceName="IDN_RDF" sessionName="Session1" />
```

5. Add the subscriptions begin and end tags between the rfa element and the closing adapter tag.

```
<subscriptions>
</subscriptions>
```

The adapter subscribes to Event Stream Processor to get the data to publish to the RMDS.

6. Between the opening and closing subscriptions tags, add opening and closing subscription tags to define a subscription. Include these attributes in the opening subscription tag:

Attribute	Description
name	Specify a unique name for this subscription.
flags	Set this parameter to "BASE" to obtain a complete set of initial values. This may be undesirable in situations such as recovery if there are a lot of unchanging values because getting those values adds latency to the other values. In these cases, set this parameter to "NO_BASE".

```
<subscription name="subscription1" flags="BASE" >
</subscription>
```

Each subscription defined in the output adapter map file must reference at least one Event Stream Processor stream.

7. Add the stream definition to the subscription.
 - a) Immediately before the closing subscription tag, insert the opening and closing stream tags. In the opening stream tag, include the name attribute set to the name of the stream.
 - b) To use a “constant” rather than a column to specify your Reuters permission code, insert the constant tag immediately before the closing stream tag, including these attributes:

Attribute	Description
name	Specify the Reuters FID “PROD_PERM.”
value	Specify the permission code issued by Reuters that certifies your permission to publish to RMDS.

- c) Immediately following the opening stream tag, insert the name tag, with the attribute column set to the column before the column with the symbol or RIC in the project. For example, if the symbol or RIC is in the first column in the project, set the value of column to 0.
 - d) Immediately following the opening name tag, insert the stale tag, with the attribute column set to one less than the position of the value in the project.
 - e) Between the stale and the constant tags, add a field tag for each data column in the stream that you want to send to RMDS. Include these attributes:

Attribute	Description
column	Set this parameter to either the name of the column or the numeric position (one less than the position of the value in the project).
name	Specify the Reuters FID for this data.

For fields of datatype float, you may also include the precision attribute, set to the number of digits you want after the decimal place in the value sent to RMDS. For example:

```
<stream name="stream1" >
<name column="0" />
<stale column="3" />
<field column="4" name="BID" precision="5" />
<field column="5" name="ASK" precision="0" />
<field column="6" name="TRDPRC_1"/>
<field column="7" name="ACVOL_1"/>
<constant name="PROD_PERM" value="1"/>
</stream>
```

Running the Output Adapter

Run the adapter once you have configured it.

Prerequisites

Configure an adapter.

Task

1. Ensure that **esp_server** is running and that the project has been loaded and started.
2. Start the adapter using the **esp_rmds** command.
 - a) If the Event Stream Processor is running with no authentication or encryption, start the adapter:

```
esp_rmds -a out -f mapfile -p cluster_host:cluster_port/
workspace/project
```

using the appropriate mapfile, cluster_host, cluster_port, workspace, and project names for the project to which the adapter will connect.

- b) If the Event Stream Processor is running with encryption or some form of authentication, refer to *Command Usage* to obtain the additional arguments necessary for the command to start the adapter.

The exact usage of the command depends on how you started your Event Stream Processor. You must invoke the adapter with compatible options. The command string shown invokes neither encryption nor authentication: you can specify either or both.

3. The adapter starts the subscription by first connecting to Event Stream Processor and then connecting to RMDS. Both connections must be operational for any data to flow.

If you plan to direct the adapter's log output to stderr, as shown here, you may want to redirect stderr to a log file (for example, append `>& myrmdslog &` to the command line shown above).

Testing the Adapter

If the adapter is not working as expected, you can perform a quick sanity check by executing the **esp_rmds** command and verifying whether the adapter is sending Reuters market data to Event Stream Processor.

- Execute **esp_rmds**:

```
esp_rmds -v
```

- This command returns the adapter release number and the revision number of the source tree separated by an underscore character. Ensure that your version of the adapter is compatible with your version of Event Stream Processor.
- There are several ways to verify whether the Reuters Marketfeed adapter is publishing to RMDS:

- Use the **tail** command on the adapter log file to which console output was redirected or any of the Reuters publisher log files (specified in `rfa.pub.cfg`) to look for activity.
- Use the **esp_subscribe** command to look at the outbound stream and verify that values are changing.
- Use RMDS tools to subscribe to RICs provided by the output adapter.
- Use an input adapter to subscribe to the output adapter.

Creating a Subordinate Map File

Create a subordinate map file to hold part of the map file configuration.

It can be advantageous to put part of your input or output adapter map file in a separate file. For example, you might want to keep a subscription configuration in a map file, but break out the list of RICs you want the adapter to subscribe to.

1. Go to the directory that contains the map file.
2. Create a new file with the extension `.xml`.
You need not add a declaration of the XML version.
3. Insert the selected content from the map file into the new file.

The content you add depends on which part of the map file you have decided to store separately.

4. (Optional) Add a comment to the new file.
5. Save the file.

Modifying the Main Map File

Modify the main map file to reference the subordinate file.

1. Make sure the first line of the main map file is:

```
<?xml version="1.0"?>
```

2. Between the XML version declaration and the opening adapter tag, add these lines:

```
<!DOCTYPE adapter SYSTEM "adapter.dtd" [  
]>
```

3. For each subordinate map file:

- a) Between the two lines just added, add:

```
<!ENTITY SUBREF SYSTEM "SUBFILE">
```

where SUBREF is a string to reference the subordinate file and SUBFILE is the path and file name of the subordinate file itself. Enclose the path and file name in quotation marks.

- b) Remove the content that you put in the subordinate map file.
- c) Insert a string like the following to include the content from the subordinate map file:

```
&SUBREF;
```

where SUBREF is the string you specified to reference the subordinate file.

Example

Configure the input adapter in the map file (`subInclude.map.xml`) to reference two subordinate files (`RIClist1.sm.mf.xml`, and `RIClist2.sm.mf.xml`).

The map file `subInclude.map.xml` configures the input adapter to reference two subordinate files, each containing a list of RICs for the adapter to subscribe to.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE adapter [
<!ENTITY RIClist1 SYSTEM "RIClist1.sm.mf.xml">
<!ENTITY RIClist2 SYSTEM "RIClist2.sm.mf.xml">
<!ENTITY rmdsFields SYSTEM "rmds.sm.mf.xml">
]>
<adapter>
<publication name="RMDS Adapter" retryInterval="5"
sendAsTransactions="0" flushInterval="1000"
intraSubscribeDelay="100"/>
<streamMaps>
<streamMap name="stream1">
&rmdsFields;
</streamMap>
</streamMaps>
<rfa config="$ESP_REUTERS_HOME/config/rmdsmf.cfg"
sessionName="Inbound" />
<itemLists>
&RIClist1;
&RIClist2;
</itemLists>
</adapter>
```

The first file, `RIClist1.sm.mf.xml`, contains:

```
<!-- This fragment is meant to be included in an itemLists section.-->
<!-- These are FX RICs -->
<itemList service="IDN_RDF" stream="stream1">
<item name="GRMN.O"/>
<item name="INTC.O"/>
<item name="KLAC.O"/>
<item name="XLNX.O"/>
<item name="YHOO.O"/>
</itemList>
```

The second file, `RIClist2.sm.mf.xml`, contains:

```
<!-- This fragment is meant to be included in an itemLists section.-->
<!-- These are FX RICs -->
<itemList service="IDN_RDF" stream="stream1">
<item name="AUD="/>
<item name="CAD="/>
<item name="DKKTN="/>
<item name="GBPSW="/>
<item name="GBPTN="/>
```

```

<item name="JPYSN=" />
<item name="JPYSW=" />
<item name="JPYTN=" />
<item name="HKD=" />
<item name="SGDSW=" />
<item name="ZAR=" />
<item name="ZARSN=" />
</itemList>

```

Performance Tuning

There are several attributes you can use to fine-tune performance in an input adapter.

Attribute	Description
<code>flushInterval</code>	Specify an interval of time in microseconds (for example, 5000 microseconds = 5 milliseconds) to wait while accumulating data. At the end of this interval, any accumulated events are sent to Event Stream Processor. Send events less often to allow more events to be placed into a message, resulting in a communications overhead savings. Use a nonzero <code>flushInterval</code> to make even accumulation time-based.
<code>maxRecordsPerBlock</code>	Specify the maximum number of accumulated events that the adapter should send to Event Stream Processor at a time. When the number of accumulated events is larger than this value, the envelope or transaction is broken into fragments that are less than or equal to the specified value. For example, if accumulated event counts of more than 1024 (which would immediately fill the Event Stream Processor Gateway's inbound queue) are expected, set <code>maxRecordsPerBlock</code> to a value like 500 to prevent the inbound queue from filling.
<code>pendingLimit</code>	Specify a threshold for the number of events that must accumulate before they are sent to Event Stream Processor. Set this parameter to zero to publish each event immediately when it happens (providing the lowest latency), at the expense of high network overhead (a TCP/IP packet for each update). If you set this parameter to a larger value, the adapter waits until number of events have accumulated, packs them efficiently in TCP/IP packets, and sends them to Event Stream Processor. This saves communication work but increases latency on both the adapter and Event Stream Processor.

Attribute	Description
sendAsTransactions	<p>This parameter controls whether events are sent as an envelope or a transaction. You can specify this parameter on a per-stream basis.</p> <p>Set this parameter to true for Event Stream Processor to treat a group of events as a single transaction. Transactions typically cause application-level workload savings, since Event Stream Processor collapses multiple events to the same value (as determined by identical key columns) in a transaction to a single event. If a transaction contains a delete, additional savings are achieved since updates prior to the delete can be discarded.</p> <p>If you set this parameter to false and you are not in low-latency mode (pendingLimit and flushInterval both set to zero), then use the maxRecordsPerBlock to control the size of the envelope. You still gain the communications overhead savings mentioned above, but not the transactional savings. This is the preferred configuration for applications that require every event to be sent separately, such as a market data compliance application.</p> <p>As a general rule, for quote-based applications, where only the most recent update matters, use transactions to be most efficient. For trades, however, every event must be processed separately to compute a total volume, use envelopes instead.</p>

When you use both flushInterval and pendingLimit, no event waits longer than the time indicated in the flushInterval before being sent, and as long as the number of events specified in pendingLimit arrive, they are sent immediately. The adapter waits flushInterval and, if any events have accumulated, it sends them. If the number of pendingLimit events, or more, accumulate while the adapter is sending the earlier events, the new events are sent immediately (without waiting for the flushInterval). If fewer than the number of pendingLimit events accumulate while the adapter is sending events, it waits for the flushInterval to elapse.

You can also use the rfaQueue attribute at the itemLists, itemList, or item element level. When specified, the rfaQueue attribute causes the element to be subscribed from Reuters on a named rfaQueue. Each rfaQueue is processed by its own thread within the Reuters adapter. Spreading requests across multiple threads can reduce latency and improve overall adapter throughput at the cost of greater CPU usage.

Since all events (images and updates) for the same RIC come from Reuters on the same queue, the integrity of the order of arrival is maintained for any individual RIC. If you do not specify an rfaQueue for any of the elements, a single default queue (named "default") is used for all RICs.

Command Usage

The Reuters Marketfeed adapter converts data from the Reuters Market Data System (RMDS) to the Event Stream Processor and vice versa.

Synopsis

```
esp_rmds -f mapFile -p host:port/workspace/project [ OPTION ...]
```

Description

esp_rmds can operate as either an input or an output adapter. An input adapter passes data from RMDS in to the Event Stream Processor. An output adapter passes data from the Event Stream Processor out to RMDS. A single adapter instance cannot operate both ways. To have an input adapter and an output adapter, you must run two separate adapter instances.

The metadata describing the connection has several parts, including a map file, a configuration file, and possibly a configuration stream resident on a running instance of the Event Stream Processor.

Only limited Level 2 data is available via RMDS Marketfeed. For full order book depth, use the Reuters OMM adapter (**esp_rmdsomm**).

The Marketfeed adapter process runs as a daemon, getting its configuration from a map file. It handles SIGHUP; so you can enter `kill -s SIGHUP pid` on Linux or `kill -s HUP pid` on Solaris (where `pid` is the process ID of the **esp_rmds** daemon, which you can obtain using the **ps** command) to gracefully shut down the adapter. Using the KILL signal rather than the HUP signal may prevent a complete clean up of system resources.

There are three directories containing additional information underneath the directory where the adapter is installed: `doc`, `examples`, and `config`. The `doc` directory contains Reuters README files that describe various configuration options. The `examples` directory contains several example map files that demonstrate many features. The `config` directory contains example RMDS configuration files. Minimally, you must modify the RMDS `config` file with your site's specific information. Typically, you must also modify the map file to match the Event Stream Processor.

Required Arguments

- **-f mapFile** – specify the map file containing the metadata required to map the market data to/from RMDS.
- **-p host:port/workspace/project** – specify the URI to connect to the server (cluster manager). For example, `-p localhost:19011/default/prj1` specifies a project called `prj1` in the default workspace of an ESP cluster server using port 19011 on the machine at which you entered the command.

Options

- **-a in|out** – specify whether the RMDS Marketfeed adapter instance is passing data in to the Event Stream Processor or receiving data passed out from it. Valid values are `in` and `out`. Since the default value is `in`, this option is typically omitted when subscribing to market data.

For backward compatibility, "subscribe" (`in`) and "publish" (`out`) are still allowed, but these options have been deprecated.

- **-c user:password** – if you are using an authentication method that requires credentials (such as Kerberos, PAM, or RSA), this option passes those authentication credentials to Event Stream Processor. If Event Stream Processor successfully authenticates with these credentials, the connection is maintained, otherwise Event Stream Processor immediately closes the connection.
- **-d debugLevel** – set the debug level. The valid range is 0 - 7, with 0 being minimal and 7 being verbose. By default, the debug level is 4.
- **-e** – negotiate encrypted OpenSSL sockets for all communication with the Event Stream Processor, which must be started in encrypted mode when using this option.
- **-F configFile** – specify the RMDS configuration file, overriding the configuration file specified in the map file.
- **-g gatewayHost** – explicitly the Event Stream Processor gateway host.
- **-G** – use Kerberos authentication. This option is required when the Event Stream Processor is started with the **-V gssapi** option.
- **-h** – print a short help message describing the syntax of this command.
- **-k privateRSAKeyFile** – perform authentication using the RSA private key file mechanism instead of password authentication. The `privateRSAKeyFile` must specify the absolute path file name of the private RSA key file. With this option enabled, the user name must be specified with the `-c` option, but the password is not required. In addition, Event Stream Processor must be started with the `-k` option.
- **-l 0|1|2|3** – specify the location to which log messages are sent. Use 0 for no log messages, 1 to send to `stderr` only (the default), 2 to send to `syslog` only, and 3 to send to both `stderr` and `syslog`.
- **-r subscribeRetryInterval** – specify how many seconds to wait (default is 300) between attempts to resubscribe to a RIC. (If a subscription to a RIC is marked `CLOSED` or `CLOSEDRECOVER`, you must resubscribe to that RIC for data to flow.) To disable resubscription attempts, specify 0 as the value. Periodically resubscribing can compensate for a temporary condition where the source is not ready for subscribers. Each unsuccessful resubscribe attempt generates a failure event which may result in a status update marking the item stale.
- **-s streamName** – specify the stream to be used when running in discovery mode. This option is used by the connector start mechanism and specifies the single stream for which mapped columns have been discovered.

CHAPTER 2: Adapters Supported by Event Stream Processor

- **-v** – print the version of the RMDS Marketfeed Adapter and exit.
- **-w retrySeconds** – specify the number of seconds to wait between retries when connecting to the Event Stream Processor. The default is 5. Specify 0 to try only once.
- **-x optName** – specify various extra settings; use `-x help` to see a list of possible values.
- **-z publishCount** – specify the number of values to pass to the Event Stream Processor before terminating. By default this is 0, which means never terminate.
- **-Z subscribeCount** – specify the number of values to pass to RMDS before terminating. By default this is 0, which means never terminate.

Examples

To start an input adapter, using the map file `subexample.map.xml`, running a project named `project1` in a workspace named `wsl` on port 19011 of the localhost machine, enter:

```
esp_rmdds -c user:pw -p localhost:19011/wsl/project1 -a in -f
subexample.map.xml
```

Environment Variables

The Reuters Marketfeed adapters use environment variables to specify behavior.

Environment Variable	Used By	Description
ESP_ACCUMULATOR_DELAY	Input	(Expert) Delay connection to the Event Stream Processor (seconds).
ESP_DISABLE_REPORT_ENCODING_NULL	Output	Stop warning about blank to null conversions (bool) [false].
ESP_FLUSH_INTERVAL	Input	Override the publication flushInterval (microseconds).
ESP_INTRASUBSCRIBE_DELAY	Input	Override the map attribute (milliseconds).
ESP_LOG_CONFIG_EVENTS	Both	Set log level (1–7) for config event processing [-1].
ESP_MARKETFEED_DUMP	Output	Set the log level (0–7) at which to dump raw Reuters messages to the log.
ESP_MAX_RECORDS_PER_BLOCK	Input	Override the publication maxRecordsPerBlock (count).
ESP_PENDING_LIMIT	Input	Override the publication pendingLimit.
ESP_RETRY_INTERVAL	Both	Override the publication retryInterval.
ESP_REUTERS_HOME	Both	Specify the installation directory.

Environment Variable	Used By	Description
ESP_RMDS_DISPATCH	Both	(Expert) Dispatch RFA every N milliseconds [10,000].
ESP_RMDS_EVENT_TRACE	Both	(Expert) Enables RFA event tracing every N event (int).
ESP_RMDS_PUBLISH_BUFSIZE	Output	Override the buffer size.
ESP_RMDS_PUBLISH_DEBUG_LEVEL	Output	Set to 7 to see values.
ESP_RMDS_PUBLISH_DEBUG_SYMBOLS	Output	Contains a space-delimited list of symbols that are used when default behavior is overridden. If this environment variable is not set, all symbols are used.
ESP_RMDS_SUBSCRIBE_DEBUG_LEVEL	Input	Set to 7 to see values.
ESP_RMDS_SUBSCRIBE_DEBUG_SYMBOLS	Input	Contains a space-delimited list of symbols for above. If this environment variable is not set, all symbols are used.
ESP_RMDS_SUBSCRIBE_SYMBOL_FORMAT	Input	Specify symbol list format: 0 for multiline; or 1 for single line.
ESP_SEND_AS_TRANSACTIONS	Input	Override the map attribute.
ESP_SHOW_FIELD_INFO	Input	Show FID, column, spColumn, and stream name [false].
ESP_SHOW_SP_EVENT_DATA	Output	Set log level (1–7) for events from the Event Stream Processor [-1].

Input Adapter Map File XML Syntax

The syntax of the map file for a Reuters Marketfeed input adapter.

```

adapter                (required, limit one)
|  ---publication      (required, limit one)
|  ---streamMaps      (required, limit one)
|    '----streamMap   (required)
|    |  ---itemName   (required, limit one)
|    |  ---serviceName (optional)
|    |  ---sequenceNumber (optional)
|    |  ---itemStale   (optional)
|    |  ---dataField   (required)
|    |  ---updateNumber (required)
|    |  ---dateTimeField (optional)

```

CHAPTER 2: Adapters Supported by Event Stream Processor

```

|-----FIDListField      (optional)
|-----nullField        (optional)
----recordTypeMap        (optional)
|-----recordType        (optional)
----rfa                   (required, limit one)
|-----itemLists         (required, limit one)
|-----itemList          (required)
|-----item              (optional)

```

adapter

The **adapter** element is the root element of the map file.

Summary

```

adapter                    (required, limit one)
|----publication           (required, limit one)
|----streamMaps           (required, limit one)
|      |----streamMap      (required)
|      |      |----itemName (required, limit one)
|      |      |----serviceName (optional)
|      |      |----sequenceNumber (optional)
|      |      |----itemStale (optional)
|      |      |----dataField (required)
|      |      |----updateNumber (required)
|      |      |----dateTimeField (optional)
|      |      |----FIDListField (optional)
|      |      |----nullField (optional)
|----recordTypeMap        (optional)
|      |----recordType      (optional)
|----rfa                   (required, limit one)
|----itemLists             (required, limit one)
|      |----itemList        (required)
|      |----item            (optional)

```

Nest all the configuration sections between the **adapter** start and end tags.

Parent

None

Children

The following child elements are defined for the adapter element. All of these elements must be in the order specified.

Name	Requirement
publication	Exactly one required
streamMaps	Exactly one required
recordTypeMap	Optional
rfa	Exactly one required

Name	Requirement
itemLists	Exactly one required

Attributes

Name	Description	Requirement
name	A string that uniquely identifies this adapter (included in log entries)	Optional

Notes

None

Example

See the examples for the individual elements contained within the **adapter** definition.

dataField

In the **streamMap** definition, the **dataField** element maps one column from a source stream to a Reuters Field ID (FID).

Summary

```

adapter                                (required, limit one)
|
|----publication                        (required, limit one)
|----streamMaps                        (required, limit one)
|   '----streamMap                    (required)
|       |----itemName                  (required, limit one)
|       |----serviceName                (optional)
|       |----sequenceNumber             (optional)
|       |----itemStale                  (optional)
|       |----dataField                  (required)
|       |----updateNumber               (required)
|       |----dateTimeField              (optional)
|       |----FIDListField               (optional)
|       |----nullField                  (optional)
|----recordTypeMap                     (optional)
|   '----recordType                    (optional)
|----rfa                                (required, limit one)
|----itemLists                          (required, limit one)
|   '----itemList                       (required)
|       '----item                       (optional)

```

Parent

streamMap

Children

None

Attributes

Name	Description	Requirement
name	The Reuters FID that identifies the data item that appears in this column of the source stream	Required
key	Either true or false, depending on whether this column is part of the source stream's unique key	See Notes

Notes

Each element in the **streamMap** section of the input adapter map file must represent a column in the row definition of the target source stream. (The order of the streamMap elements must mirror the order of the columns in the RowDef.) If the column in the RowDef is a data item (Bid, Ask, and so on), the corresponding **streamMap** entry must be a **dataField** element for which the name attribute identifies a specific FID. Any time RMDS publishes an update tagged with that FID, the adapter sends it to Event Stream Processor source stream as a value in the corresponding row.

Use the **key** attribute to set the value to true. If this column is not part of the stream's key, you can omit the key attribute.

Example

```
<streamMap name="stream1">
  <itemName key="true"/>
  <FIDListField />
  <!-- serviceName / -->
  <sequenceNumber />
  <itemStale/>
  <dataField name="BID"/>
  <dataField name="ASK"/>
  <dataField name="TRDPRC_1"/>
  <dataField name="ACVOL_1"/>
  <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>
```

The example shown above maps columns 5–8 of stream1 to the Reuters FIDs BID, ASK, TRDPRC_1, and ACVOL_1.

dateTimeField

In the **streamMap** definition, the **dateTimeField** element maps a Reuters date or time FID (or one of each) to a date column, a timestamp column, or both, in a stream.

Summary

```
adapter          (required, limit one)
| ----publication (required, limit one)
| ----streamMaps (required, limit one)
|   '----streamMap (required)
```



```

|-----itemName      (required, limit one)
|-----serviceName  (optional)
|-----sequenceNumber (optional)
|-----itemStale     (optional)
|-----dataField     (required)
|-----updateNumber  (required)
|-----dateTimeField (optional)
|-----FIDListField  (optional)
|-----nullField     (optional)
|-----recordTypeMap (optional)
|-----recordType    (optional)
|-----rfa           (required, limit one)
|-----itemLists     (required, limit one)
|-----itemList      (required)
|-----item          (optional)

```

Parent

streamMap

Children

None

Attributes

Name	Description	Requirement
dateName	The FID of the date value provided by RMDS	See Notes
timeName	The FID of the time value provided by RMDS	See Notes

Notes

The most commonly used datatype for date/time information in Event Stream Processor data streams is `dateTime`, which combines both date and time. In most cases, however, the updates provided by RMDS and brought in to the Event Stream Processor by the Reuters Marketfeed adapter use separate FIDs for date and time.

To address this discrepancy, the map file provides the `dateTimeField` element, which provides separate attributes for date and time, allowing you to map two FIDs (one for date, one for time) to the same column in the stream definition.

If `dateTime` is used, it must be used alone. The `dateName` and `timeName` can be used either separately or together. One of these three attributes must be used.

The value for each FID must match one listed in the FID list referenced in the Reuters-side configuration file (the FID list provided with the adapter is named `appendix_a`). This file is referenced in the configuration file `rfa.sub.cfg`.

Example

```

<streamMap name="stream1">
  <itemName key="true"/>

```

```

        <FIDListField />
        <!-- serviceName / -->
        <sequenceNumber />
        <itemStale/>
        <dataField name="BID" />
        <dataField name="ASK" />
        <dataField name="TRDPRC_1" />
        <dataField name="ACVOL_1" />
        <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE" />
    </streamMap>

```

This example maps the TIMACT and ACTIV_DATE FIDs together to the ninth column of the Event Stream Processor source stream named stream1.

FIDListField

In the **streamMap** definition, the **FIDListField** element maps all of the Reuters FIDs with their values for an event to the Event Stream Processor source stream.

Summary

```

adapter                                (required, limit one)
  |----publication                      (required, limit one)
  |----streamMaps                      (required, limit one)
  |   |----streamMap                  (required)
  |   |   |----itemName              (required, limit one)
  |   |   |----serviceName          (optional)
  |   |   |----sequenceNumber       (optional)
  |   |   |----itemStale             (optional)
  |   |   |----dataField             (required)
  |   |   |----updateNumber         (required)
  |   |   |----dateTimeField        (optional)
  |   |   |----FIDListField        (optional)
  |   |   |----nullField            (optional)
  |   |----recordTypeMap             (optional)
  |   |   |----recordType           (optional)
  |   |----rfa                       (required, limit one)
  |   |----itemLists                 (required, limit one)
  |   |   |----itemList             (required)
  |   |   |   |----item             (optional)

```

Parent

streamMap

Children

None

Attributes

Name	Description	Requirement
name	A string that will appear in any adapter-related log entries	Optional

Notes

None

Example

```

<streamMap name="stream1">
  <itemName key="true"/>
  <FIDListField />
  <!-- serviceName / -->
  <sequenceNumber />
  <itemStale/>
  <dataField name="BID"/>
  <dataField name="ASK"/>
  <dataField name="TRDPRC_1"/>
  <dataField name="ACVOL_1"/>
  <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>

```

In this example, the second column of the source stream is identified as the one that carries the FIDList string of any update from the adapter.

item

The **item** element is used to identify a RIC to which the Reuters Marketfeed adapter subscribes.

Summary

```

adapter                (required, limit one)
  ----publication      (required, limit one)
  ----streamMaps      (required, limit one)
    '----streamMap    (required)
      |----itemName    (required, limit one)
      |----serviceName (optional)
      |----sequenceNumber (optional)
      |----itemStale    (optional)
      |----dataField    (required)
      |----updateNumber (required)
      |----dateTimeField (optional)
      |----FIDListField (optional)
      |----nullField    (optional)
    ----recordTypeMap (optional)
      '----recordType  (optional)
    ----rfa            (required, limit one)
    ----itemLists      (required, limit one)
      '----itemList    (required)
        '----item      (optional)

```

Parent

itemList

Children

None

Attributes

Name	Description	Requirement
name	An RIC to which the adapter subscribes	Required
rfaQueue	A name for the rfaQueue, which, if provided, replaces the default rfaQueue name and causes separate thread to be used for this queue	Optional
service	The name of a Reuters Service that provides incoming data through RMDS	Optional if already specified in the parent itemList or itemLists element, otherwise required
stream	The source stream on which updates for this RIC are brought to the Event Stream Processor	Optional if already specified in the parent itemList or itemLists element, otherwise required

Notes

The value for the name attribute must match one listed in the `appendix_a` file referenced in the Reuters-side configuration file (`rfaSub.cfg` is the name of the file provided with the adapter).

If you specify a stream name here, updates for this RIC are brought in to the Event Stream Processor on that stream. If you do not specify a stream here, the stream specified at the `itemList` level is used.

The stream you specify must match one of the `streamMaps` defined elsewhere in the map file by the value of the `streamMap`'s name attribute.

Example

```
<itemLists service="SSL_PUB" stream="stream1">
  <itemList service="IDN_RDF" >
    <item name="EUR=" />
    <item name="EURJPY=" stream="stream6" />
  </itemList>
</itemLists>
```

These two **item** elements subscribe the adapter to the RICs EUR and EURJPY. The EUR updates are sent to the `stream1`, which is set in the `itemLists` element. The EURJPY updates are sent to `stream6`, since the **item** level stream attribute overrides the **itemLists** level attribute.

itemList

The **itemList** element contains one or more instances of the **item** element.

Summary

```

adapter (required, limit one)
  ----publication (required, limit one)
  ----streamMaps (required, limit one)
    '----streamMap (required)
      |----itemName (required, limit one)
      |----serviceName (optional)
      |----sequenceNumber (optional)
      |----itemStale (optional)
      |----dataField (required)
      |----updateNumber (required)
      |----dateTimeField (optional)
      |----FIDListField (optional)
      |----nullField (optional)
  ----recordTypeMap (optional)
    '----recordType (optional)
  ----rfa (required, limit one)
  '----itemLists (required, limit one)
    '----itemList (required)
      '----item (optional)
    
```

Parent

itemLists

Children

Name	Requirement
item	Zero or more required

Attributes

Name	Description	Requirement
rfaQueue	A name for the rfaQueue which, if provided, replaces the default rfaQueue name and causes a separate thread to be used for this queue	Optional
service	The name of a Reuters Service that provides incoming data through RMDS	Optional if already specified in the parent itemLists element or in all child item elements, otherwise required

Name	Description	Requirement
stream	The name of an Event Stream Processor source stream receives updates on the RICs specified in this list of items	Optional if already specified in the parent itemLists element or in all child item elements, otherwise required

Notes

Configure the adapter to push updates for every item in this section to the specified stream. However, you can override the stream specification at the item level.

The adapter supports more than one item element under itemLists; this allows you to configure one instance of the adapter to direct updates from two or more groups of RICs to different Event Stream Processor source streams.

The stream you specify must match, by the value of the name attribute, one of the streamMaps defined elsewhere in the map file.

Use the rfaQueue attribute to control scalability.

Example

```
<itemLists service="SSL_PUB" stream="stream1">
  <itemList service="IDN_RDF" >
    <item name="EUR=" />
    <item name="EURJPY=" stream="stream6" />
  </itemList>
</itemLists>
```

This item element sets the service attribute to IDN_RDF, overriding the SSL_PUB service attribute defined in the parent itemLists element.

itemLists

The **itemLists** element contains one or more instances of the **itemList** element.

Summary

```
adapter (required, limit one)
|----publication (required, limit one)
|----streamMaps (required, limit one)
|   '----streamMap (required)
|       |----itemName (required, limit one)
|       |----serviceName (optional)
|       |----sequenceNumber (optional)
|       |----itemStale (optional)
|       |----dataField (required)
|       |----updateNumber (required)
|       |----dateTimeField (optional)
|       |----FIDListField (optional)
|       |----nullField (optional)
|----recordTypeMap (optional)
```

```

| '----recordType          (optional)
|----rfa                  (required, limit one)
|----itemLists            (required, limit one)
|      '----itemList      (required)
|      '----item          (optional)

```

Parent
adapter

Children

Name	Requirement
itemList	One required, two or more supported

Attributes

Name	Description	Requirement
name	A string that appears in any adapter-related log entries	Optional
rfaQueue	A name for the rfaQueue which, if provided, replaces the default rfaQueue name and causes a separate thread to be used for this queue	Optional
service	The name of a Reuters service that provides incoming data through RMDS	Optional if specified in the child itemLists or item elements so that all child item elements either specify or inherit it, otherwise required
stream	The name of an Event Stream Processor source stream that receives updates on the RICs specified in the item lists in this section (a default that can be overridden at the item level)	Optional if specified in the child itemLists and/or item elements so that all child item elements either specify or inherit it, otherwise required

Notes

Each **itemList** instance in this section is a list of one or more of the RICs to which the adapter subscribes.

Example

```

<itemLists service="SSL_PUB" stream="stream1">
  <itemList service="IDN_RDF" >
    <item name="EUR=" />
    <item name="EURJPY=" stream="stream6" />
  </itemList>
</itemLists>

```

CHAPTER 2: Adapters Supported by Event Stream Processor

This `itemLists` element sets the `service` attribute to `SSL_PUB` and the `stream` attribute to `stream1`. These attributes are either inherited or overridden at the `itemList` and/or `item` level.

itemName

In the `streamMap` definition, the `itemName` element identifies the row in the Event Stream Processor source stream that carries the RIC from the RMDS update.

Summary

```

adapter                (required, limit one)
  |
  |----publication      (required, limit one)
  |----streamMaps      (required, limit one)
  |   |----streamMap    (required)
  |   |   |----itemName (required, limit one)
  |   |   |----serviceName (optional)
  |   |   |----sequenceNumber (optional)
  |   |   |----itemStale (optional)
  |   |   |----dataField (required)
  |   |   |----updateNumber (required)
  |   |   |----dateTimeField (optional)
  |   |   |----FIDListField (optional)
  |   |   |----nullField (optional)
  |   |----recordTypeMap (optional)
  |   |   |----recordType (optional)
  |   |----rfa           (required, limit one)
  |   |----itemLists    (required, limit one)
  |   |   |----itemList (required)
  |   |   |   |----item (optional)
  
```

Parent

streamMap

Children

None

Attributes

Name	Description	Requirement
key	True or false, depending on whether or not this column is part of the source stream's unique key	See first note

Notes

You do not need to use the `key` attribute. It is present for backward compatibility.

Insert the `itemName` element in the `streamMap` to correspond with the column in the `RowDef` that carries the RIC or symbol. If this column is part of the source stream's key, set the `key` attribute to true.

This element is one of the "pseudofields" that specify data items that are not part of the data feed coming directly from RMDS.

Example

```
<streamMap name="stream1">
  <itemName key="true"/>
  <FIDListField />
  <!-- serviceName / -->
  <sequenceNumber />
  <itemStale/>
  <dataField name="BID"/>
  <dataField name="ASK"/>
  <dataField name="TRDPRC_1"/>
  <dataField name="ACVOL_1"/>
  <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>
```

The first column of the stream is identified as the one that carries the RIC value of any update from the adapter. It is also identified as part of the stream's key.

itemStale

In the **streamMap** definition, the **itemStale** element identifies a column in the Event Stream Processor source stream that carries a flag indicating whether incoming RMDS data has gone stale.

Summary

```
adapter (required, limit one)
  ----publication (required, limit one)
  ----streamMaps (required, limit one)
    '----streamMap (required)
      ----itemName (required, limit one)
      ----serviceName (optional)
      ----sequenceNumber (optional)
      ----itemStale (optional)
      ----dataField (required)
      ----updateNumber (required)
      ----dateTimeField (optional)
      ----FIDListField (optional)
      ----nullField (optional)
    ----recordTypeMap (optional)
      '----recordType (optional)
    ----rfa (required, limit one)
    ----itemLists (required, limit one)
      '----itemList (required)
        '----item (optional)
```

Parent

streamMap

Children

None

Attributes

Name	Description	Requirement
name	A string that appears in any adapter-related log entries	Required

Notes

Use this element in the streamMap if one of the columns in the source stream is a "stale" flag.

RMDS itself does not supply a stale flag with regular market data, although it may pass along such a flag if it is provided by another service you are subscribing to via RMDS. If this element is used in the streamMap, the adapter sends out an update value of 1 if it receives a stale flag from RMDS, or stops receiving any data from RMDS.

Example

```
<streamMap name="stream1">
  <itemName key="true"/>
  <FIDListField />
  <!-- serviceName / -->
  <sequenceNumber />
  <itemStale/>
  <dataField name="BID"/>
  <dataField name="ASK"/>
  <dataField name="TRDPRC_1"/>
  <dataField name="ACVOL_1"/>
  <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE" />
</streamMap>
```

The fourth column of the source stream is identified as the one that is updated if the adapter receives a stale notification, or stops receiving data from RMDS.

nullField

In a **streamMap**, the **nullField** element acts as a placeholder that always delivers a NULL value to the stream. This lets you add extra fields to a source stream to get the configuration you want.

Summary

```
adapter                (required, limit one)
  |----publication      (required, limit one)
  |----streamMaps      (required, limit one)
  |      '----streamMap (required)
  |          |----itemName (required, limit one)
  |          |----serviceName (optional)
  |          |----sequenceNumber (optional)
  |          |----itemStale (optional)
```

```

|-----dataField      (required)
|-----updateNumber  (required)
|-----dateTimeField (optional)
|-----FIDListField  (optional)
|-----nullField     (optional)
|-----recordTypeMap (optional)
|-----recordType    (optional)
|-----rfa           (required, limit one)
|-----itemLists     (required, limit one)
|-----itemList      (required)
|-----item          (optional)

```

Parent

streamMap

Children

None

Attributes

Name	Description	Requirement
name	A string that appears in any adapter-related log entries	Optional

Notes

When experimenting with a project, you can use a **nullField** to temporarily stop feeding data into one column of the stream. In this case, you can simply keep the name of the **dataField** that you are temporarily replacing, as in the following example.

Example

```

<streamMap name="stream1">
  <itemName key="true"/>
  <FIDListField />
  <!-- serviceName / -->
  <sequenceNumber />
  <itemStale />
  <dataField name="BID"/>
  <dataField name="ASK"/>
  <nullField name="TRDPRC_1"/>
  <dataField name="ACVOL_1"/>
  <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>

```

The seventh column of the source stream is identified as a placeholder receives a null value in each update from the adapter. It includes the name of the **dataField** that it replaces for debugging purposes.

publication

The **publication** element specifies basic operating parameters for this instance of the adapter.

Summary

```

adapter (required, limit one)
  ----publication (required, limit one)
  ----streamMaps (required, limit one)
    '----streamMap (required)
      |----itemName (required, limit one)
      |----serviceName (optional)
      |----sequenceNumber (optional)
      |----itemStale (optional)
      |----dataField (required)
      |----updateNumber (required)
      |----dateTimeField (optional)
      |----FIDListField (optional)
      |----nullField (optional)
  ----recordTypeMap (optional)
    '----recordType (optional)
  ----rfa (required, limit one)
  '----itemLists (required, limit one)
    '----itemList (required)
      '----item (optional)
    
```

Parent

adapter

Children

None

Attributes

Name	Description	Requirement
flushInterval	Specify the number of microseconds the adapter allows events to accumulate before sending them to the Event Stream Processor. A non zero flushInterval makes event accumulation time-based.	Optional
intraSubscribeDelay	Specify the number of milliseconds the adapter pauses between subscription requests.	Optional

Name	Description	Requirement
maxRecordsPerBlock	Specify the maximum number of accumulated events that the adapter should send to the Event Stream Processor at a time. This reduces the size of each transaction or envelope fragment when there is a large number of accumulated events. For example, if 140 events have accumulated and maxRecordsPerBlock is set to 50, the adapter will send the envelope or transaction as three fragments.	Optional
name	Specify a string that identifies the adapter instance in log file entries.	Required
pendingLimit	Specify the number of events that may accumulate before the adapter sends them in to the Event Stream Processor. Using a pendingLimit makes event accumulation count-based.	Optional
retryInterval	Specify the number of seconds for which the adapter attempts to connect to RMDS before shutting down.	Required
sendAsTransactions	Set to true to treat a group of updates as a single transaction or false to treat them as separate transactions within an envelope.	Optional

Notes

You can optimize the adapter's performance using the **pendingLimit** and **flushInterval** attributes, along with the **maxRecordsPerBlock** and **sendAsTransactions** attributes from the Pub/Sub interface that the adapter uses to communicate with the Event Stream Processor.

Some venues send initial images as multi part messages, which may produce large data sets. The **intraSubscribeDelay** attribute paces these subscriptions and prevents the adapter from being overwhelmed by initial images. The default value is zero, which is suitable for short RIC lists. When **intraSubscribeDelay** is set to a nonzero value, the adapter pauses between subscription requests for milliseconds. The suggested value is ten (10).

Example

```
<publication name="RMDS Adapter - low latency" retryInterval="5"
  flushInterval="0" pendingLimit="0" sendAsTransactions="0" />
```

recordType

The **recordType** element maps a stream to a predefined set of FIDs.

Summary

```
adapter (required, limit one)
  |----publication (required, limit one)
```

```

----streamMaps (required, limit one)
  '----streamMap (required)
    |----itemName (required, limit one)
    |----serviceName (optional)
    |----sequenceNumber (optional)
    |----itemStale (optional)
    |----dataField (required)
    |----updateNumber (required)
    |----dateTimeField (optional)
    |----FIDListField (optional)
    |----nullField (optional)
  ----recordTypeMap (optional)
    '----recordType (optional)
  ----rfa (required, limit one)
  ----itemLists (required, limit one)
    '----itemList (required)
      '----item (optional)

```

Parent

recordTypeMap

Children

None

Attributes

Name	Description	Requirement
number	The ID of a recordType defined in Reuters configuration	Required
stream	The name of a stream to which this record will be mapped	Required

Notes

The pre-defined record specified by **recordType** must match all the columns in the stream's definition. Otherwise, these columns must be explicitly mapped in a **streamMap** configuration.

Example

```

<recordTypeMap>
  <recordType number="123" stream="eqInput"/>
</recordTypeMap>

```

This example maps a set of FIDs pre-defined as record "123" to the source stream eqInput.

recordTypeMap

The **recordTypeMap** element contains one or more instances of **recordType**.

Summary

```

adapter (required, limit one)
  |----publication (required, limit one)

```

```

----streamMaps (required, limit one)
  '----streamMap (required)
    |----itemName (required, limit one)
    |----serviceName (optional)
    |----sequenceNumber (optional)
    |----itemStale (optional)
    |----dataField (required)
    |----updateNumber (required)
    |----dateTimeField (optional)
    |----FIDListField (optional)
    |----nullField (optional)
  ----recordTypeMap (optional)
    '----recordType (optional)
  ----rfa (required, limit one)
  ----itemLists (required, limit one)
    '----itemList (required)
      '----item (optional)

```

Parent
adapter

Children

Name	Requirement
recordType	Zero or more supported

Attributes
None

Notes

A stream must have either a recordTypeMap or a streamMap; it cannot have both.

The pre-defined record must match all the columns in the stream's definition to use the implicit mapping provided by recordTypeMap. Otherwise, these columns must be explicitly mapped in a streamMap configuration.

Example

```

<recordTypeMap>
  <recordType number="123" stream="eqInput" />
</recordTypeMap>

```

This example maps a set of FIDs pre-defined as record "123" to the source stream eqInput.

rfa

The **rfa** element links the input adapter map file to the Reuters-side configuration file.

Summary

```

adapter (required, limit one)
  ----publication (required, limit one)
  ----streamMaps (required, limit one)
    '----streamMap (required)
      |----itemName (required, limit one)
      |----serviceName (optional)
      |----sequenceNumber (optional)
      |----itemStale (optional)
      |----dataField (required)
      |----updateNumber (required)
      |----dateTimeField (optional)
      |----FIDListField (optional)
      '----nullField (optional)
  ----recordTypeMap (optional)
    '----recordType (optional)
  ----rfa (required, limit one)
  ----itemLists (required, limit one)
    '----itemList (required)
      '----item (optional)
    
```

Parent

adapter

Children

None

Attributes

Name	Description	Requirement
config	The absolute path and file name of the Reuters-side configuration file for subscription (the sample file supplied with the adapter is \$ESP_REUTERS_HOME/config/rfasub.cfg).	Required
configDatabaseName	Must be set to RFA.	Required
enumFile	The full path name of the Reuters-supplied file that lists each enumerated type along with the range of values it can take.	Required
fidFile	The full path name of the Reuters-supplied file that lists all of the valid FIDs.	Required

Name	Description	Requirement
sessionName	A reference to a session name defined in the Reuters-side configuration file for subscription.	Required
blank	Specify a marker to use for blanks	Optional
blankInt32	Specify a marker to use for blank Int32 fields	Optional
blankInt64	Specify a marker to use for blank Int64 fields	Optional
blankMoney	Specify a marker to use for blank Money fields	Optional
blankString	Specify a marker to use for blank String fields	Optional
blankDate	Specify a marker to use for blank Date fields	Optional
blankTimestamp	Specify a marker to use for blank Timestamp fields	Optional

Notes

None

Example

```
<rfa config="$ESP_REUTERS_HOME/config/rfasub.cfg"
  sessionName="Session1" />
```

This example points the Reuters Marketfeed adapter to the Reuters-side configuration in the file `rfasub.cfg`. The list line in this configuration file is:

```
\Sessions\Session1\connectionList =
"Connection_SSLED"
```

This line defines a session name that is referenced by other lines in the configuration file. When the map file references a session name in the **sessionName** attribute, it links the adapter to the Reuters-side configuration parameters identified by that name.

sequenceNumber

In the **streamMap** definition, the **sequenceNumber** element maps a column in the source stream that is populated by a unique number generated by the adapter, not provided as part of the data from RMDS.

Summary

```
adapter (required, limit one)
  |----publication (required, limit one)
  |----streamMaps (required, limit one)
  |   |----streamMap (required)
  |   |   |----itemName (required, limit one)
  |   |   |----serviceName (optional)
  |   |   |----sequenceNumber (optional)
  |   |   |----itemStale (optional)
```

```

|         |----dataField      (required)
|         |----updateNumber  (required)
|         |----dateTimeField (optional)
|         |----FIDListField (optional)
|         |----nullField  (optional)
|----recordTypeMap      (optional)
|   '----recordType     (optional)
|----rfa                 (required, limit one)
|----itemLists           (required, limit one)
|   '----itemList       (required)
|     '----item         (optional)

```

Parent
streamMap

Children
None

Attributes

Name	Description	Requirement
key	True or false, depending on whether this column is part of the source stream's unique key	See Notes
name	A string that appears in log entries	Optional

Notes

The adapter maintains a separate counter for each RIC to which it is subscribed. Each time it receives an update for a RIC, it increments the counter for that RIC. This number is the one sent to the source stream column mapped by the `sequenceNumber` element.

Many source stream definitions include a column specification similar to:

```
<Column datatype="int32" name="Id" />
```

This line specifies a unique ID for the source stream. The **sequenceNumber** pseudo field is a good match for this column in the input adapter map file

You must use the key attribute to set the value to true. If this column is not part of the stream's key, you can omit this.

Example

```

<streamMap name="stream1">
  <itemName key="true"/>
  <FIDListField />
  <!-- serviceName / -->
  <sequenceNumber />
  <itemStale/>
  <dataField name="BID"/>
  <dataField name="ASK"/>

```

```
<dataField name="TRDPRC_1" />
<dataField name="ACVOL_1" />
<dateTimeField timeName="TIMACT" dateName="ACTIV_DATE" />
</streamMap>
```

The third column of the source stream is mapped to the sequence number provided by the adapter. This column is also identified as part of the source stream's unique key.

serviceName

In the **streamMap** definition, the **serviceName** element maps a column in the source stream to the service identifier that the adapter provides as part of the envelope for each update.

Summary

```
adapter (required, limit one)
  |----publication (required, limit one)
  |----streamMaps (required, limit one)
  |   |----streamMap (required)
  |   |   |----itemName (required, limit one)
  |   |   |----serviceName (optional)
  |   |   |----sequenceNumber (optional)
  |   |   |----itemStale (optional)
  |   |   |----dataField (required)
  |   |   |----updateNumber (required)
  |   |   |----dateTimeField (optional)
  |   |   |----FIDListField (optional)
  |   |   |----nullField (optional)
  |----recordTypeMap (optional)
  |   |----recordType (optional)
  |----rfa (required, limit one)
  |----itemLists (required, limit one)
  |   |----itemList (required)
  |   |   |----item (optional)
```

Parent

streamMap

Children

None

Attributes

Name	Description	Requirement
key	True or false, depending on whether this column is part of the source stream's unique key	See Notes

Notes

You must use the key attribute to set the value to true. If this column is not part of the stream's key, you can omit this.

Example

```

<streamMap name="stream1">
  <itemName key="true"/>
  <FIDListField />
  <!-- serviceName / -->
  <sequenceNumber />
  <itemStale />
  <dataField name="BID"/>
  <dataField name="ASK"/>
  <dataField name="TRDPRC_1"/>
  <dataField name="ACVOL_1"/>
  <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>

```

In this example, no column of the source stream is mapped to the service name provided by the adapter because it is commented out.

streamMap

The **streamMap** element contains the mappings between the columns of an Event Stream Processor source stream and the RMDS FIDs being subscribed to by the adapter.

Summary

```

adapter                (required, limit one)
  ----publication      (required, limit one)
  ----streamMaps      (required, limit one)
    '----streamMap    (required)
      ----itemName    (required, limit one)
      ----serviceName (optional)
      ----sequenceNumber (optional)
      ----itemStale    (optional)
      ----dataField    (required)
      ----updateNumber (required)
      ----dateTimeField (optional)
      ----FIDListField (optional)
      '----nullField  (optional)
  ----recordTypeMap    (optional)
    '----recordType   (optional)
  ----rfa              (required, limit one)
  ----itemLists        (required, limit one)
    '----itemList     (required)
      '----item       (optional)

```

Parent

streamMaps

Children

The following child elements are defined for **streamMap**. These child elements can occur in any order, but for a specific **streamMap**, the order of the child elements must mirror the order

CHAPTER 2: Adapters Supported by Event Stream Processor

of the columns of the source stream (as defined in the project). This is how the adapter is configured to deliver RMDS updates to the appropriate rows in the source stream.

Name	Requirement
dataField	One required, two or more supported
dateTimeField	Zero or more supported
itemName	One required, two or more supported
itemStale	Zero or one supported
sequenceNumber	Zero or one supported
serviceName	Zero or one supported

Attributes

Name	Description	Requirement
name	References the source stream to which the RMDS updates are mapped. Must match the name of a source stream defined in the Event Stream Processor project.	Required
opcode	Defines the operation the adapter performs when sending updates to the source stream. Possible values are insert and upsert. The insert operation adds new updates to the end of the source stream. The upsert operation replaces an existing source stream entry if its key matches the entry's key; if not, the update is added.	Optional (default value is upsert)

Notes

None

Example

```
<streamMaps>
  <streamMap name="stream1">
    <itemName key="true"/>
    <FIDListField />
    <!-- serviceName / -->
    <sequenceNumber />
    <itemStale/>
    <dataField name="BID"/>
    <dataField name="ASK"/>
    <dataField name="TRDPRC_1"/>
    <dataField name="ACVOL_1"/>
    <dateTimeField timeName="TIMACT"
dateName="ACTIV_DATE"/>
  </streamMap>
</streamMaps>
```

CHAPTER 2: Adapters Supported by Event Stream Processor

This example maps a set of the adapter's updates to an Event Stream Processor source stream named stream1. All updates going to this source stream are added using the upsert mode. The RICs for which updates are sent to this source stream are specified in an itemList elsewhere in the map file that also references stream1.

streamMaps

The **streamMaps** element contains one or more instances of the **streamMap** element.

Summary

```
adapter                (required, limit one)
  ----publication      (required, limit one)
  ----streamMaps       (required, limit one)
    '----streamMap     (required)
      |----itemName    (required, limit one)
      |----serviceName (optional)
      |----sequenceNumber (optional)
      |----itemStale    (optional)
      |----dataField    (required)
      |----updateNumber (required)
      |----dateTimeField (optional)
      |----FIDListField (optional)
      |----nullField    (optional)
  ----recordTypeMap    (optional)
    '----recordType    (optional)
  ----rfa               (required, limit one)
  '----itemLists        (required, limit one)
    '----itemList      (required)
      '----item         (optional)
```

Parent

adapter

Children

Name	Requirement
streamMap	One required, two or more supported

Attributes

None

Notes

Each **streamMap** instance in this section maps incoming FIDs from the Reuters adapter to columns in an Event Stream Processor source stream.

Example

```
<streamMaps>
  <streamMap name="stream1">
    <itemName key="true"/>
```

```

        <FIDListField />
        <!-- serviceName / -->
        <sequenceNumber />
        <itemStale />
        <dataField name="BID" />
        <dataField name="ASK" />
        <dataField name="TRDPRC_1" />
        <dataField name="ACVOL_1" />
        <dateTimeField timeName="TIMACT"
dateName="ACTIV_DATE" />
    </streamMap>
</streamMaps>

```

updateNumber

In the **streamMap** definition, the **updateNumber** element maps a column in the Event Stream Processor source stream that is populated by a unique number generated by the adapter, not provided as part of the data from RMDS.

Summary

```

adapter (required, limit one)
  |----publication (required, limit one)
  |----streamMaps (required, limit one)
  |   |----streamMap (required)
  |   |   |----itemName (required, limit one)
  |   |   |----serviceName (optional)
  |   |   |----sequenceNumber (optional)
  |   |   |----itemStale (optional)
  |   |   |----dataField (required)
  |   |   |----updateNumber (required)
  |   |   |----dateTimeField (optional)
  |   |   |----FIDListField (optional)
  |   |   |----nullField (optional)
  |----recordTypeMap (optional)
  |   |----recordType (optional)
  |----rfa (required, limit one)
  |----itemLists (required, limit one)
  |   |----itemList (required)
  |       |----item (optional)

```

Parent

streamMap

Children

None

Attributes

Name	Description	Requirement
key	True or false, depending on whether this column is part of the source stream's unique key	See Notes
name	A string that appears in log entries	Optional

Notes

The adapter maintains a separate counter for each RIC to which it is subscribed. Each time it receives an update for a RIC, it increments the counter for that RIC. This number is the one sent to the source stream column mapped by the **updateNumber** element.

Many source stream definitions include a column specification similar to:

```
<Column datatype="integer" name="Id"/>
```

This line specifies a unique ID for the source stream. The **updateNumber** pseudo field is a good match for this column in the input adapter map file.

You must use the key attribute to set the value to true. If this column is not part of the stream's key, you can omit this attribute.

Example

```
<streamMap name="stream1">
  <itemName key="true"/>
  <FIDListField />
  <!-- serviceName / -->
  <updateNumber />
  <itemStale/>
  <dataField name="BID"/>
  <dataField name="ASK"/>
  <dataField name="TRDPRC_1"/>
  <dataField name="ACVOL_1"/>
  <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>
```

The third column of the source stream is mapped to the update number provided by the adapter. This column is also identified as part of the source stream's unique key.

Output Adapter Map File XML Syntax

The syntax of the map file for a Reuters Marketfeed output adapter.

The following listing shows the structure of an output adapter map file. Each line of this summary lists one element of the map file structure. See the topics for each element for details.

```
adapter (required, limit one)
  |----rfa (required, limit one)
  '----subscriptions (required, limit one)
    '----subscription (required)
```



```

      '----stream          (required)
      |----name           (required, limit one)
      |----service        (optional)
      |   '----enum       (required)
      |----stale          (optional)
      |----field          (required)
      '----constant       (optional)
    
```

adapter

The **adapter** element is the root element of the output map file.

Summary

```

adapter          (required, limit one)
|----rfa         (required, limit one)
'----subscriptions (required, limit one)
   '----subscription (required)
      '----stream     (required)
      |----name       (required, limit one)
      |----service    (optional)
      |   '----enum   (required)
      |----stale      (optional)
      |----field      (required)
      '----constant   (optional)
    
```

Nest all configuration elements between the start and end **adapter** tags.

Parent

None

Children

The following child elements are defined for **adapter**. All of these elements must be present in the specified order.

Name	Requirement
rfa	Exactly one required
subscriptions	Exactly one required

Attributes

None

Notes

None

Example

See the examples for the child elements.

constant

The **constant** element defines a data item with a constant value that is published to RMDS by the adapter.

Summary

```

adapter (required, limit one)
|----rfa (required, limit one)
|----subscriptions (required, limit one)
|       '----subscription (required)
|           '----stream (required)
|               |----name (required, limit one)
|               |----service (optional)
|               |       '----enum (required)
|               |----stale (optional)
|               |----field (required)
|               '----constant (optional)
    
```

Parent

stream

Children

None

Attributes

Name	Description	Requirement
name	The name associated with this data item in the image published by the adapter	Required
value	The value of this constant (always the same whenever this data item is published to RMDS)	Required

Notes

At start-up, the adapter publishes a complete image, containing all data items defined in the map file, to RMDS. After that, the adapter publishes updated values for data items when they change, unless the Event Stream Processor goes stale and then recovers. This means that the value for **constant** is published only when a complete image is published.

Example

```

<stream name="stream1" >
  <name column="0" />
  <stale column="3" name="ACVOL_1" />
  <field column="1" name="DSPLY_NAME" />
  <field column="4" name="BID" precision="47" />
  <field column="5" name="ASK" precision="0" />
  <field column="6" name="TRDPRC_1" />
  <field column="7" name="ACVOL_1" />
    
```

```
<constant name="PROD_PERM" value="1"/>
</stream>
```

This example defines a constant called `PROD_PERM`, with the constant value 1, to be published with data values from the stream1 under the publication name subscription1.

enum

The **enum** element maps the value of the Event Stream Processor stream's service column to a unique string that is prepended to the name element of an update published to RMDS by the adapter.

Summary

```
adapter (required, limit one)
  |----rfa (required, limit one)
  '----subscriptions (required, limit one)
    '----subscription (required)
      '----stream (required)
        |----name (required, limit one)
        |----service (optional)
        |   '----enum (required)
        |----stale (optional)
        |----field (required)
        |----constant (optional)
```

If the Event Stream Processor stream from which you are publishing handles data items for the same symbol from different sources (the "Ask" price for IBM from NASDAQ and from S&P, for example), you can use the `service` and `enum` attributes in the output adapter map file to configure the adapter to differentiate between updates of the same value for the same symbol from different sources.

Parent

service

Children

None

Attributes

Name	Description	Requirement
value	A possible value for the data stream column specified by the service element	Required
prefix	The string prepended to the value of the name element when it publishes updates received from the Event Stream Processor with the service value that matches prefix	Required

Notes

The **service** element in the output adapter map file must contain one **enum** element for each possible value in the source column.

Example

```
<service column="2" delim="_">
  <enum value="RDF" prefix="R"/>
  <enum value="ISFS" prefix="I"/>
</service>
```

Within a service definition, each **enum** element specifies a particular service. Based on this value, the published RICs are renamed to indicate the provider of the data. Assume that RIC.X is the RIC found in the name column. If the value in column 2 is RDF, the RIC becomes "R_RIC.X". If the value in column 2 is ISFS, the RIC becomes "I_RIC.X". If neither is true, no value is published.

field

In a **stream** definition in an output adapter map file, **field** specifies a column in a stream to publish.

Summary

```
adapter (required, limit one)
|----rfa (required, limit one)
'----subscriptions (required, limit one)
  '----subscription (required)
    '----stream (required)
      |----name (required, limit one)
      |----service (optional)
      |   '----enum (required)
      |----stale (optional)
      |----field (required)
      '----constant (optional)
```

Parent

stream

Children

None

Attributes

Name	Description	Requirement
column	A number that represents the position of the source column in the stream being published from (the first column in the stream has the number 0)	Required

Name	Description	Requirement
name	The FID that identifies this data value when published to RMDS	Required
precision	An integer that specifies the total number of digits after the decimal point in the published value (for example, 1.23 has a precision of 2)	Optional

Notes

Modify the value of the name attribute to indicate the source of the data item if you have defined the **parname** and **enum** elements in this stream definition.

Include the precision attribute only for columns of datatype double.

Example

```
<stream name="stream1" >
  <name column="0" />
  <stale column="3" name="ACVOL_1" />
  <field column="1" name="DSPLY_NAME" />
  <field column="4" name="BID" precision="47" />
  <field column="5" name="ASK" precision="0" />
  <field column="6" name="TRDPRC_1" />
  <field column="7" name="ACVOL_1" />
  <constant name="PROD_PERM" value="1" />
</stream>
```

The adapter is configured to publish updates from the fourth, fifth, sixth and seventh columns of the Event Stream Processor stream named stream1 as data items named BID, ASK, TRDPRC_1 and ACVOL_1, respectively.

name

In a **stream** definition in an output adapter map file, **name** specifies the column in the source stream that provides the value to use to identify each update.

Summary

```
adapter (required, limit one)
  |----rfa (required, limit one)
  '----subscriptions (required, limit one)
    '----subscription (required)
      '----stream (required)
        |----name (required, limit one)
        |----service (optional)
        | '----enum (required)
        |----stale (optional)
        |----field (required)
        '----constant (optional)
```

Parent

stream

Children

None

Attributes

Name	Description	Requirement
column	A number that represents the position of the column in the stream that carries the stream's unique identifier (the first column in the stream is number 0)	Either column or name
name	The name of the column in the stream that carries the stream's unique identifier	Either column or name

Notes

The output adapter uses RMDS as a simple message bus; the published updates need not conform to Reuters protocols. This means that the column specified by this element does not have to be a Reuters RIC, but it must follow Reuters RIC syntax.

If the source stream's unique key is a composition of two or more columns, you can use the name element in combination with one or more instances of the service element to configure the adapter to publish updates with completely unique names.

Example

```
<stream name="stream1" >
  <name column="0" />
  <stale column="3" name="ACVOL_1" />
  <field column="1" name="DSPLY_NAME" />
  <field column="4" name="BID" precision="47" />
  <field column="5" name="ASK" precision="0" />
  <field column="6" name="TRDPRC_1" />
  <field column="7" name="ACVOL_1" />
  <constant name="PROD_PERM" value="1" />
</stream>
```

This example identifies the first column of stream1 as its unique identifier or "key" column.

rfa

The **rfa** element provides information for configuring the Reuters side of the adapter, including an explicit reference to the Reuters-side configuration file.

Summary

```
adapter (required, limit one)
  |----rfa (required, limit one)
  '----subscriptions (required, limit one)
    '----subscription (required)
      '----stream (required)
        |----name (required, limit one)
```

```

|----service      (optional)
|  '----enum      (required)
|----stale        (optional)
|----field        (required)
|----constant     (optional)

```

Parent
adapter

Children
None

Attributes

Name	Description	Requirement
serviceName	Defines a service name that is included in the header of every update sent out by the Reuters Marketfeed adapter	Optional
config	The absolute path and file name of the Reuters-side configuration file for publication (the sample file supplied with the adapter is \$ESP_REUTERS_HOME/config/rfapub.cfg)	Required
sessionName	A reference to a session named defined in the Reuters-side configuration file for publication	Required
configDatabaseName	A reference to the Reuters database name	Optional

Notes
None

Example

```

<rfa serviceName="IDN_RDF"
    config="$ESP_REUTERS_HOME/config/rfapub.cfg"
    sessionName="Session1" configDatabaseName="RFA" />

```

This example points the Reuters Marketfeed adapter to the Reuters-side configuration in the file `rfapub.cfg`. The first four uncommented lines in this configuration file are:

```

\Connections\Connection_SSLED_MP\ipcServerName = "8105"
\Connections\Connection_SSLED_MP\connectionType = "SSLED_MP"
\Connections\Connection_SSLED_MP\entitlementData = false
\Sessions\Session1\connectionList = "Connection_SSLED_MP"

```

The last of these lines implicitly defines a session name that is defined as the **sessionName** in the map file. The other three lines from `rfapub.cfg` key on this session name. This is how

CHAPTER 2: Adapters Supported by Event Stream Processor

the value for **serviceName** ties this **publication** section of the map file to a configuration set in the `.cfg` file.

When the adapter publishes using this configuration, each update is identified with the **serviceName** "IDN_RDF".

service

In a **stream** definition in an output adapter map file, **service** identifies a column in the source stream that is another component of the stream's unique key.

Summary

```
adapter (required, limit one)
|----rfa (required, limit one)
|----subscriptions (required, limit one)
|----subscription (required)
|----stream (required)
|----name (required, limit one)
|----service (optional)
|----enum (required)
|----stale (optional)
|----field (required)
|----constant (optional)
```

Parent

stream

Children

None

Attributes

Name	Description	Requirement
column	A number that represents the position of the column with the secondary key value (the first column in the stream has the number 0)	Required
delim	Specifies a character to use as the separator between a name and a prefix	Optional

Notes

The **service** element in the output adapter map file must contain one **enum** element for each possible value in the source column.

Example

```
<service column="2" delim="_">
  <enum value="RDF" prefix="R"/>
  <enum value="ISFS" prefix="I"/>
</service>
```


This section configures the adapter to test the value of the second column of every update from the Event Stream Processor stream (the value of the **name** attribute of the **stream** element).

If the value is RDF, the adapter adds the prefix "R" followed by the specified delim value to the name of the published update (the value of the **name** attribute of the **publication** element).

If the value is ISFS, the adapter adds the prefix "I" to the **name** of the published update.

stale

In a **stream** definition in an output adapter map file, the **stale** element identifies a column in the source stream for which the value changes from 0 to 1 if the stream goes stale.

Summary

```

adapter (required, limit one)
|----rfa (required, limit one)
|----subscriptions (required, limit one)
|   '----subscription (required)
|       '----stream (required)
|           |----name (required, limit one)
|           |----service (optional)
|           |   '----enum (required)
|           |----stale (optional)
|           |----field (required)
|           '----constant (optional)
    
```

A stream is considered to have gone stale if, for example, one of the stream's data sources is no longer being updated.

Parent

stream

Children

None

Attributes

Name	Description	Requirement
column	A number representing the position of the column with the secondary key value (the first column in the stream has the number 0)	Required
name	A string that identifies the stale column so that it may be mapped to a FID (published)	Optional

Notes

None

Example

```
<stream name="stream1" >
  <name column="0" />
  <stale column="3" name="ACVOL_1" />
  <field column="1" name="DSPLY_NAME" />
  <field column="4" name="BID" precision="47" />
  <field column="5" name="ASK" precision="0" />
  <field column="6" name="TRDPRC_1" />
  <field column="7" name="ACVOL_1" />
  <constant name="PROD_PERM" value="1" />
</stream>
```

This example identifies the third column of stream1 as its stale column. If the stale column is specified, the column value is published and the RIC is marked stale.

stream

In a subscription section in an output adapter map file, identifies the stream from which the adapter obtains the data it publishes to RMDS.

Summary

```
adapter (required, limit one)
|----rfa (required, limit one)
'----subscriptions (required, limit one)
  '----subscription (required)
    '----stream (required)
      |----name (required, limit one)
      |----service (optional)
      |   '----enum (required)
      |----stale (optional)
      |----field (required)
      '----constant (optional)
```

Parent

subscription

Children

Name	Requirement
Name	One
Service	Optional
Stale	Optional
Field	One or more
Constant	Optional

Attributes

Name	Description	Requirement
exitOnStreamExit	This is a boolean attribute. When true, RMDS terminates if the stream exits, the Event Stream Processor exits, or the connection is lost.	Optional
finalizer	This string specifies an action to take if the specified number of heartbeat milliseconds elapse without an event being published to the Event Stream Processor.	Optional
heartbeat	This integer specifies the number of milliseconds to wait without an event being published to the Event Stream Processor before executing the finalizer action.	Optional
name	The name of the stream from which the adapter receives the data it publishes on RMDS	Required
templateNumber	A Reuters template set up in the RMDS configuration	Optional

Notes

You must define the value of the **name** attribute in the Event Stream Processor project.

Any stream in the Event Stream Processor project can map to only one **stream** section in the map file.

The **templateNumber** must be a unique identifier of the stream for which it is defined

Example

```
<stream name="stream1">
  <name column="0" />
  <field column="4" name="TRDPRC_1" />
  <field column="9" name="BID" precision="5" />
</stream>
```

This example configures the Event Stream Processor to publish data from a stream named stream1.

subscription

The **subscription** element contains one or more instances of the **stream** element, enabling you to configure the adapter to receive data from one or more streams.

Summary

```
adapter (required, limit one)
  |----rfa (required, limit one)
  '----subscriptions (required, limit one)
    '----subscription (required)
      '----stream (required)
        |----name (required, limit one)
```

```

|-----service      (optional)
|      '-----enum  (required)
|-----stale       (optional)
|-----field       (required)
|-----constant    (optional)

```

The output adapter map file can contain two or more **subscription** sections. At runtime, the publishing mechanism for each **subscription** section is instantiated on a separate thread, which provides scalability.

Parent

subscriptions

Children

Name	Requirement
stream	One or more

Attributes

Name	Description	Requirement
name	A name for this subscription that appears in updates published on RMDS and in log file entries	Required

Notes

None

Example

```

<subscriptions>
  <subscription name="subscription1" >
    <stream name="stream1" >
      <name column="0"/>
      <field column="4" name="BID"/>
      <field column="5" name="ASK"/>
      <field column="6" name="TRDPRC_1"/>
      <field column="7" name="ACVOL_1"/>
      <constant name="PROD_PERM" value="1"/>
    </stream>
  </subscription>
</subscriptions>

```

This example configures the adapter to publish some columns from stream1 using the name subscription1.

subscriptions

The **subscriptions** element contains one or more **subscription** elements.

Summary

```

adapter (required, limit one)
|----rfa (required, limit one)
|----subscriptions (required, limit one)
|       '----subscription (required)
|           '----stream (required)
|               |----name (required, limit one)
|               |----service (optional)
|               |       '----enum (required)
|               |----stale (optional)
|               |----field (required)
|               '----constant (optional)

```

Parent

adapter

Children

Name	Requirement
Subscription	One or more

Attributes

None

Notes

Each **subscription** instance in this section defines one set of data that the adapter publishes to RMDS.

Example

See the example for an individual **subscription** instance.

Logging Facilities

The Reuters Marketfeed adapter supports two different logging mechanisms.

In addition to its own logging mechanism, the Reuters Marketfeed adapter can utilize Reuters-side logging. You can use both of these mechanisms to check the adapter's performance and diagnose problems.

You can configure these logs to be written to stderr, syslog, or both.

Adapter Logging

The Reuters Marketfeed adapter supports the same options for logging as the Event Stream Processor.

The **-d** option sets the debug level (0=emergency messages only, 7=all messages).

The **-l** option tells the adapter to write log messages to stderr, syslog, both, or neither. If you use the **-l** option to direct adapter log messages to stderr, you may also want to redirect stderr to a file.

The name attribute of the **publication** element in the input adapter map file specifies a descriptive text string that is logged to help identify how the adapter was configured. For example, lines 3–6 of `subexample.xml` specify the **publication** element for a subscribing instance of the Reuters Marketfeed adapter, as follows:

```
<publication
  name="RMDS Adapter exp"
  retryInterval="5"
/>
```

As the adapter connects with and interacts with Event Stream Processor, this configuration causes the adapter to write log messages similar to:

```
(0.123) @1 INFO: Configuring publication with name RMDS Adapter exp
```

The first two fields are the timestamp (in seconds since start-up) and the thread number, respectively. The base time for the timestamp, along with other information, is written to the log file on startup as shown in the following example. To convert the timestamp to a date and time, simply add the number of seconds to the base time.

```
(63359098041.768) @1 NOTICE:Base time is 10/08/08-17:27:21
(0.001) @1 NOTICE:insta-a sub -c cimtest:-- -d 7
-f /home/sybase/support/1.0.3/ReutersAdapter/quotes.map.xml
-l 1 -p tigris:12192 -P 1
(0.001) @1 NOTICE:pid=28649
(0.001) @1 DEBUG:Using ESP_RMDS_SUBSCRIBE_DEBUG_LEVEL=711/
i86pc_64_spro/bin/rmds version:
1.0.3a-alpha_r18674M
```

Page Data and Partial Page Updates

Some Reuters data comes as pages which use Marketfeed partial format. Each page consists of multiple lines; initially sent as a snapshot. Page data is supported without any special configuration. The following extract from an adapter log file shows the delivery of the initial page image (which is displayed).

```
(27.729) @6 INFO:Publishing VOD.mGBPd 21 of 21 on stream1 as UPSERT
_ITEM_NAME_ STRING: VOD.mGBPd
_SERVICE_NAME_ STRING: IDN_RDF
_SEQUENCE_NUMBER_ INTEGER: 1
_ITEM_STALE_ INTEGER: 0
ROW80_1 STRING: VOD.mGBPd SI Quote Publication
ROW80_2 STRING:
```

```

ROW80_3 STRING: DATE:03/07/2008 Time:11:09
ROW80_4 STRING:
ROW80_5 STRING: Time Venue SI Bid Size Bid Price Ask Price Ask Size
Status
ROW80_6 STRING: ==== ===== == ===== ===== =====
=====
ROW80_7 STRING: 110937 GSILGB2XXXX GSIL 1 150.9000 150.9500 1 OPEN
ROW80_8 STRING: 070021 SBILGB2LXXX CITI OPEN
ROW80_9 STRING: 110909 CSFBGB2LXXX CSFB 329 150.7000 151.1500 329
OPEN
ROW80_10 STRING: 110942 DEUTGB22ZEQ DBBL 528 150.6500 151.2000 527
OPEN
ROW80_11 STRING: 110946 ABNAGB22XXX ABNV 483306 150.9000 150.9500
483306 OPEN
ROW80_12 STRING: 110936 UBSWGB2LEQU UBSI 1 149.7682 152.1325 1 OPEN
ROW80_13 STRING: 110828 SBUKGB21XXX CITI 20600 150.9000 151.0000
20600 OPEN
ROW80_14 STRING: 110937 SLIIGB2LXXX LEHM 3750 150.9000 150.9500 15
OPEN
ROW80_15 STRING:
ROW80_16 STRING:
ROW80_17 STRING:
(27.730) @6 DEBUG:Immediate flush for low latency; opcode=p

```

Each line of the page has its own FID to facilitate line-oriented deltas to the page. The adapter parses the partial page updates from Reuters and produces strings like the ones shown in the following extract from an adapter log file.

```

(49.934) @6 DEBUG:Processing update for VOD.mGBPd from service
IDN_RDF
(49.934) @6 INFO:Publishing VOD.mGBPd 4 of 21 on stream1 as UPSERT
_ITEM_NAME_ STRING: VOD.mGBPd
_SEQUENCE_NUMBER_ INTEGER: 2
ROW80_3 STRING: off:78 size:2 value:10
ROW80_11 STRING: off:2 size:3 value:101
(49.934) @6 DEBUG:Immediate flush for low latency; opcode=p
(50.315) @6 DEBUG:Processing update for VOD.mGBPd from service
IDN_RDF
(50.315) @6 INFO:Publishing VOD.mGBPd 3 of 21 on stream1 as UPSERT
_ITEM_NAME_ STRING: VOD.mGBPd
_SEQUENCE_NUMBER_ INTEGER: 3
ROW80_11 STRING: off:5 size:1 value:7
(50.315) @6 DEBUG:Immediate flush for low latency; opcode=p

```

The first update in the example is to write the 2-character string 10 at an offset of 78 characters in the line of the page which contains the data from the ROW80_3 FID. The second update in the example is to write the 3-character string 101 at an offset of 2 characters in the line of the page which contains the data from the ROW80_11 FID. The third update in the example is to write the 1-character string 7 at an offset of 5 characters in the line of the page which contains the data from the ROW80_11 FID. Thus, updates for page data are very concise.

Modifying Log Entry Format

You can modify the default format of log entries in two ways.

CHAPTER 2: Adapters Supported by Event Stream Processor

Set the environment variable `ESP_RMDS_SUBSCRIBE_SYMBOL_FORMAT` to 1 to configure your system to log messages that show what values flow to the Event Stream Processor on a single line rather than the default multi line format. When messages are written to a log file, this can make it easier to scan for specific items.

Use the `-P` option to the `esp_rmds` command to specify specify the number of decimal places that appear on output for double type variables.

By default, log messages that show what values flow to the Event Stream Processor are written in multi line format as shown.

```
(38079.526) @2 INFO:Publishing VOD.mGBPd 3 of 9 on stream1 as UPSERT
_ITEM_NAME_ STRING: VOD.mGBPd
_SEQUENCE_NUMBER_ INTEGER: 953
ROW80_7 STRING: off:53 size:2 value:45
```

If you set the environment variable `ESP_RMDS_SUBSCRIBE_SYMBOL_FORMAT` to 1 these messages are written in single-line format.

```
(17.794) @5 DEBUG:stream1 p values: _ITEM_NAME_=VOD.mGBPd
_SEQUENCE_NUMBER_=2
ROW 80_3=off:78 size:2 value:20
```

The `-P` option can alter the manner in which double datatype variables appear, as shown by ask and last are in the following example. This affects only the way variables appear; it does not alter the contents.

```
<RowDefinition id="marketfeed_RowDef">
<Column name="symbol" datatype="string" />
<Column name="service" datatype="string" />
<Column name="seq" datatype="integer" />
<Column name="stale" datatype="integer" />
<Column name="bid" datatype="money" />
<Column name="ask" datatype="double" />
<Column name="last" datatype="double" />
<Column name="volume" datatype="integer" />
<Column name="when" datatype="timestamp" />
</RowDefinition>
```

If you accept the default precision, variables of type double (for example, ASK in the following example) are written with three digits to the right of the decimal

```
(5.089) @5 INFO:Publishing EURJPY= 7 of 9 on stream1 as UPSERT
(5.090) @5 DEBUG:stream1 p values: _ITEM_NAME_=EURJPY=
_SEQUENCE_NUMBER_=1 _ITEM_STALE_=0 BID=137.4800 ASK=137.530
ACVOL_1=0
ACTIV_DATE+TIMACT=2008-10-06T21:07:00.000 (1223327220000)
```

If you specify the option `-P 7` when enter the `esp_rmds` command, variables of type double (for example, ASK in the following example) are written with seven digits to the right of the decimal. Variables of other types are not affected.

```
(4.913) @5 INFO:Publishing EURJPY= 7 of 9 on stream1 as UPSERT
(4.913) @5 DEBUG:stream1 p values: _ITEM_NAME_=EURJPY=
_SEQUENCE_NUMBER_=1 _ITEM_STALE_=0 BID=137.5200 ASK=137.5700000
ACVOL_1=0 ACTIV_DATE+TIMACT=2008-10-06T20:55:00.000 (1223326500000)
```


Reuters Logging

Turn Reuters logging on or off using the Reuters-side configuration file.

You can configure the adapter's interface to RMDS to write to a logging facility. In the Reuters-side configuration file (`rfasub.cfg` and `rfapub.cfg` are the ones provided with the adapter), you can turn logging on or off and specify a path and file name of the log file. The Reuters interface also supports a set of "message files."

The Reuters-side configuration file contains a set of configuration entries for the Reuters "Logger" facility.

```

\Logger\AppLogger\fileLoggerEnabled           = true
\Logger\AppLogger\fileLoggerFilename         = "rfasub.{p}.log"

```

This configuration turns on Reuters logging for the Reuters Marketfeed adapter. The log messages are written to the `rfasub.PID.log` file, where **PID** is the adapter's process ID.

The first line in this set, `\Logger\AppLogger\windowsLoggerEnabled = false`, pertains to a Windows logging facility that is not supported for the Reuters Marketfeed adapter.

These example lines are from `rfasub.cfg`, the file that configures an adapter that subscribes to RMDS. The configuration file for publication, `rfapub.cfg`, contains the same configuration lines (except that the value for **fileLoggerFilename** is `rfapub.{p}.log`).

The same file contains configuration entries for Component Loggers, as follows:

```

\Logger\ComponentLoggers\Connections\messageFile = "config/
messages/RFA7_Connections.mc"
\Logger\ComponentLoggers\Adapter\messageFile     = "config/
messages/RFA7_Adapter.mc"
\Logger\ComponentLoggers\SessionCore\messageFile = "config/
messages/RFA7_SessionLayer.mc"
\Logger\ComponentLoggers\SSLED_Adapter\messageFile = "config/
messages/RFA7_SSLED_Adapter.mc"

```

Log Messages

Examples of typical entries from the adapter log file.

The actual format and working of the log messages, as well as the nature of the events logged and the log levels associated with these events, may change in subsequent releases of the adapter.

- **Message:** – NOTICE:Item BARC.VX is closed: No Quality of Service is available to process subscription, timeout expired
- **Cause:** – the value for the Reuters user name in the Reuters config file is incorrect (verify the case-sensitivity) or the Reuters Service name in the map file is incorrect.
- **Message:** – DEBUG: Immediate flush for low latency

- **Cause:** – data received from RMDS is being sent to Event Stream Processor immediately.
- **Message:** – NOTICE:XMLRPC ERROR-116: The connection to the server could not be established. Please make sure the server is up, and check the specified host name/port, user name/password, and encryption settings. If a host name is specified, make sure that it can be resolved through a DNS lookup. (5.092) @1 INFO:Could not connect to SP; (tigris:12190 cimtest) will retry in 5 seconds.
- **Cause:** – cannot connect to the server running Event Stream Processor.
- **Message:** – Ignoring market data event because no significant fields updated
- **Cause:** – the adapter received data from Reuters, but none of the fields were of interest to Event Stream Processor stream, so no data was sent.
- **Message:** – ERROR: Error publishing: PUBLICATION ERROR-442: The send method of this publication object failed.
- **Cause:** – connection to Event Stream Processor unsuccessful during a message transmission.
- **Message:** – ERROR:Mismatch between platform stream (9 columns) and adapter (31 columns for stream: stream1)
- **Cause:** – the number of columns defined in the adapter did not match the number of columns in the stream.
- **Message:** – WARNING: Event Stream Processor down, dropping all subscriptions
followed by multiple iterations of a message similar to:
DEBUG: Unsubscribing item: EUR= service: IDN_RDF
- **Cause:** – lost connection to Event Stream Processor. Stopping subscriptions to RMDS data since the adapter has nowhere to put it.
- **Message:** – WARNING: Discarding data rec'd after unsubscribe
- **Cause:** – before the adapter shut off the subscription, additional data arrived. The data has been discarded because there is no connection to Event Stream Processor.
- **Message:** – DEBUG: Processing update for EUR= from service IDN_RDF
- **Cause:** – an update for RIC "EUR=" on service named "IDN_RDF" has arrived.
- **Message:** – WARNING: Event Stream Processor down, dropping all subscriptions
followed by numerous repetitions of:

DEBUG: Unsubscribing item: EUR= service: IDN_RDF

- **Cause:** – lost connection to Event Stream Processor. Stopping subscriptions to RMDS data since the adapter has nowhere to put it.
- **Message:** – WARNING: Discarding data rec'd after unsubscribe
- **Cause:** – before the adapter shut off the subscription, additional data arrived. The data has been discarded, because there is no connection to Event Stream Processor.
- **Message:** – EMERGENCY: Fatal Error at line 0, column 0 of config file: An exception occurred! Type:RuntimeException, Message:The primary document entity could not be opened. Id=/home/sybase/adapter/trunk/src/ReutersAdapter/xxsubexample.xml
- **Cause:** – specified configuration file is unavailable.
- **Message:** – EMERGENCY: Fatal Error at line 0, column 0 of config file: An exception occurred! Type:RuntimeException, Message:The primary document entity could not be opened. Id=/home/sybase/adapter/trunk/src/ReutersAdapter/xxsubexample.xml
- **Cause:** – specified config file is unavailable.

Reuters OMM Adapter

The Sybase Event Stream Processor Reuters OMM adapter is a software interface between Event Stream Processor and the Reuters Market Data System (RMDS). It uses the Reuters Open Message Model (OMM) message format.

You can configure the adapter as an input or output adapter. The input adapter subscribes to one or more Reuters Instrument Codes (RICs) on the RMDS to provide input to Event Stream Processor. The output adapter publishes output from Event Stream Processor to the RMDS. This enables Event Stream Processor to use the speed and reliability of Reuters' infrastructure to deliver data.

The Reuters OMM Input adapter supports schema discovery. Run two adapter instances if you require both input and output capabilities.

The adapter runs only on Solaris and Linux operating systems but you can use it with Event Stream Processor software running on Solaris, Linux, or Windows.

Requirements

The Reuters OMM input and output adapters have several requirements.

An input adapter requires:

- An RMDS market data connection that uses the Reuters Open Message Model (OMM) protocol

CHAPTER 2: Adapters Supported by Event Stream Processor

- A working subscription for data on one or more financial instruments

An output adapter requires:

- A working connection with support for sending data to RMDS using the OMM protocol

General Configuration

Enable user access for each user account that runs the Reuters OMM adapter, and configure an input connection from Reuters and an output connection to Reuters.

Enabling User Access

Enable user access for each user account that uses the Reuters OMM adapter.

1. Ensure the user account has permission to execute the installed software.
2. Create an environment variable, `$ESP_RMDSOMM_HOME`, and set to the full path name of the directory in which you placed the adapter distribution file.
3. (Optional) Add the environment variable to your shell profile.
4. Event Stream Processor supports RSA, Kerberos, and LDAP authentication. If your installation uses one of these authentication methods, ensure the user account is set up to work with that method of authentication.

Configuring an Input Connection from Reuters

Modify the sample configuration file for your site's RMDS connection. If you have multiple adapters using multiple RMDS connections, you may need a separate and uniquely named configuration file for each one. For a configuration file with a different name, either change the entry in the input adapter map file or specify that file name using the `-f` option to the `esp_rmdsomm` command.

Prerequisites

- Create (or choose) a directory in which to store your site-specific configuration files.
- Create an environment variable (`MY_CONFIG`) and set it to the full path name of that directory.

Task

During the installation process, a sample configuration file (`rmdsomm.cfg`) was placed in the `$ESP_RMDSOMM_HOME/config` directory. This file follows the Reuters format for configuration files and includes this section for your site-specific information:

```
##
## Site-specific values for OMM Inbound - subscribing from RMDS
##
\Connections\Connection_RSSL\connectionType = "RSSL"
### Caution: post value comments like below confuse RFA parsing
causing coredump
#\Connections\Connection_RSSL\hostName      = "localhost" ## not
here
```

```

\Connections\Connection_RSSL\hostName      = "localhost"
\Connections\Connection_RSSL\rsslPort     = "14002"
\Connections\Connection_RSSL\connectRetryInterval = 7000
\Sessions\Session1\connectionList        = "Connection_RSSL"

```

1. Obtain this information from your system administrator:
 - Name of the server from which you receive RMDS OMM data
 - Port number on that machine to which your system connects
 - Name of the Reuters service to which you subscribe
2. Make a copy of the sample configuration file in your `$MY_CONFIG` directory.


```
cp $ESP_RMDSOMM_HOME/config/rmdsomm.cfg $MY_CONFIG
```
3. Use a text editor to open the configuration file.
4. In the `\Connections\Connection_RSSL\rsslPort` line, replace the default port number (14002) with the port used by your Reuters connection, if different.
5. In the `\Connections\Connection_RSSL\hostName` line, replace `tigris.sybase.com` with the name of your server that receives OMM data from RMDS (keep the surrounding quotation marks).

If your system has more than one server receiving data from RMDS, include all of their names in a comma-separated list, in priority order.
6. (Optional) In the `\Logger\AppLogger\fileLoggerFilename` line, change the name of the log file.

The default file name `rfasub.{p}.log`, includes the string `{p}` which the Reuters library replaces with the UNIX process ID when it creates the log file.
7. Save the modified file.

The other parameters in the configuration file also affect the functioning of the Reuters OMM adapter, and you may want to modify them as well.

Configuring an Output Connection to Reuters

Modify the sample configuration file for your site's RMDS connection. If you have multiple adapters using multiple RMDS connections, you may need a separate and uniquely named configuration file for each one. For a configuration file with a different name, either change the entry in the output adapter map file or specify that file name using the `-F` option to the `esp_rmdsomm` command.

Prerequisites

- Create (or choose) a directory in which to store your site-specific configuration files.
- Create an environment variable (`MY_CONFIG`) and set it to the full path name of that directory.

Task

During the installation process a sample configuration file, `rmdsomm.cfg`, was placed in the `$ESP_RMDSOMM_HOME/config` directory. This file follows the Reuters format for

CHAPTER 2: Adapters Supported by Event Stream Processor

configuration files, and includes sections for site-specific information for noninteractive and interactive publishing to RMDS.

1. Obtain this information from your system administrator:
 - Port number at which the src_dist or RMDS infrastructure server listens for updates from the Reuters OMM adapter
 - Name of the server that receives updates from Event Stream Processor
2. Decide whether to publish to RMDS interactively or non-interactively.
3. If you have not already done so when specifying an input connection from Reuters, make a copy of the sample configuration file in your \$MY_CONFIG directory.

```
cp $ESP_RMDSOMM_HOME/config/rmdsomm.cfg $MY_CONFIG
```

4. Use a text editor to open the configuration file.
 - a) If you are going to publish to RMDS interactively, go to the site-specific information section for interactive publishing. In the \Connections \Connection_RSSL_PROV\connectionType line, refer to the value “RSSL_PROV,” which is the Reuters term for an information provider.

```
##
## Site-specific values for OMM Outbound - Interactive
## publishing to RMDS
##
# Interactive publisher
\Connections\Connection_RSSL_PROV\connectionType = "RSSL_PROV"
## grab a free port until the MDH is setup with 2nd src_dist
instance
\Connections\Connection_RSSL_PROV\rsslPort = "14007"
\Connections\Connection_RSSL_PROV\connectRetryInterval = 7000
\Connections\Connection_RSSL_PROV\hostName =
"tigris.sybase.com"
\Sessions\SessionOMMProv\connectionList =
"Connection_RSSL_PROV"
```

In the \Connections\Connection_RSSL_PROV\rsslPort line, replace the default port number (14007) with the port number at which your IPC server listens for updates from the Reuters OMM adapter, if different.

- b) If you are going to publish to RMDS non-interactively, go to the site-specific information section for noninteractive publishing. In the \Connections \Connection_RSSL_CPROV\connectionType line, refer to the value “RSSL_CPROV,” which is the Reuters term for a client provider.

```
##
## Site-specific values for OMM Outbound - Non-interactive
## publishing to RMDS
##
# non-interactive publisher
\Connections\Connection_RSSL_CPROV\connectionType =
"RSSL_CPROV"
\Connections\Connection_RSSL_CPROV\hostName =
"tigris.sybase.com"
## Within Sybase, this non-standard port is a proxy to the
```

```
standard 14003
\Connections\Connection_RSSL_CPROV\rsslPort = "14010"
\Connections\Connection_RSSL_CPROV\connectRetryInterval = 7000
\Sessions\SessionOMMCPProv\connectionList =
"Connection_RSSL_CPROV"
```

In the `\Connections\Connection_RSSL_CPROV\rsslPort` line, replace the default port number (14010) with the port number at which your IPC server listens for updates from the Reuters OMM adapter, if different.

5. To change the name of the log file, go to the local file logging section.

```
##
## General values
##
## local file logging
\Logger\AppLogger\windowsLoggerEnabled = false
\Logger\AppLogger\fileLoggerEnabled = true
\Logger\AppLogger\fileLoggerFilename = "rfa.{p}.log"
```

In the `\Logger\AppLogger\fileLoggerFilename` line, replace the default name, `rfapub.{p}.log`, with the name you want to use. The Reuters library replaces the `{p}` string in the default file name with the UNIX Process ID when it creates the log file.

6. Save the modified file.

Input Adapter Configuration

Configure an input adapter to push data from the Reuters Market Data Service (RMDS) to Event Stream Processor.

Before configuring an input adapter, decide what data you need and how you want to set up your system.

You need to know the following about the Event Stream Processor instance from which you receive data.

- Possible security options in a cluster environment, and the workspace and project name.
- What type of authentication mechanism (Kerberos, RSA, LDAP, or none) does it use?

Data Decisions

Decide how the incoming Reuters data fits into the project.

Also decide whether you require Level 1 or Level 2 data. For Level 2 data, use the OMM Adapter, and for Level 1 data, you can use either OMM MarketPrice messages or the Reuters Marketfeed adapter.

Decision	Description
Venues	Decide which venues are of interest (for example, NYSE, NAS-DAQ, Toronto, and so on).

CHAPTER 2: Adapters Supported by Event Stream Processor

Decision	Description
RICs and FIDs	Determine what market data you need. Specifically, which Reuters Instrument Codes (RICs) you want the adapter to provide to Event Stream Processor, and which Reuters Field IDs (FIDs) for these instruments you want to use.
Streams	The Reuters adapter can furnish data to one or more streams on Event Stream Processor. To use the Reuters Market Data provided by the adapter, decide which existing data streams to map to the adapter's data feed or define one or more new streams.

Administrative Decisions

You have several administrative decisions to make in regards to the project.

Decision	Description
Session Name	An arbitrary string used to link the project and the adapter map file. Use the session name consistently. The adapter supports only one session per adapter instance.
Directories for logging and stream output	The adapter writes its own log messages and can generate a separate set of Reuters log messages. In the configuration, specify whether and where to write these log files.
Sybase user account	Specify a valid Event Stream Processor user account for the adapter to use, unless you specified no authentication when you started the Event Stream Processor.

Input Adapter Map File

The map file configures the interface between the Reuters OMM adapter and Event Stream Processor. It specifies which source streams receive data from RMDS via the adapter, and it maps specific RMDS Field Identifiers (FIDs) to specific columns in that source stream.

The input adapter map file must accomplish two major tasks:

- Match incoming data elements to columns in one or more streams defined in the Event Stream Processor configuration file.
- Ensure that each update from the adapter can be converted into a record that provides a unique key for each stream being populated, as defined by the stream's column definitions.

Data Structures

Data structures have three important structural aspects: data columns, datatypes, and key values.

- Each data stream includes one or more data columns.
- Each column has a datatype.

- Each row has a unique key value. The source stream definition designates one or more columns as "key" columns. Data must be fed to a source stream.

Incoming RMDS Data

When the adapter subscribes to RMDS for a certain RIC, RMDS first sends an initial image containing all available market data for that RIC. After that, RMDS sends an update only when any values for a subscribed RIC change.

Each FID defined for RMDS has a datatype.

Market Data Field Mapping

Map each column in the target Event Stream Processor stream to a Reuters FID or a "pseudofield."

Find the appropriate FID for each column in the stream. The datatype of the Event Stream Processor column must be compatible with the datatype of the Reuters FID that feeds it.

Here are possible matches between FID datatypes and Event Stream Processor datatypes:

Event Stream Processor Datatype	Reuters Datatype
integer	enumeration, time_seconds
integer or long	uint32
long	uint64
money, float, integer, or long	real32
money or float	real64
string	ASCII_string, RTES_string
date or timestamp	date, time

Note: OMM supports milliseconds as part of a time field. When mapping to or from a timestamp column, milliseconds are preserved.

Reuters Instrument Code Mapping

The identifier of each incoming RMDS update is the Reuters Instrument Code (RIC).

Map the RIC to a column of datatype `string` in the stream. If the stream you want to map to does not have a suitable column, either add a column to the stream or map to a different stream.

Matching the Stream's Key

The adapter map file must configure the adapter so that every update sent to the Event Stream Processor stream includes a field or combination of fields that conform to the unique key defined for that stream. To make this more flexible, the adapter configuration mechanism supports "pseudofields."

The market data updates that the adapter receives from RMDS are mapped to columns in the Event Stream Processor stream using the dataField or dateTimeField element in the map file. RMDS also provides nonmarket data information and each update includes a RIC. Additionally, you can configure the adapter to add a sequence number to each update.

To make these data items available to the mapping process, the map file mechanism supports these elements called "pseudofields."

Field	Description	Datatype
dataField	Values such as PRICE, SIZE	(Required) datatype is determined at runtime from FIDs and stream schema)
dateTimeField	The date and time	(optional) string
itemName	The RIC	(required) string
imageField	Flag to indicate if an entry is an image	(required for Level 2 data) integer
itemStale	The item state	(optional)integer
marketByOrder-KeyField	Secondary key for Level 2 messages	(required for Level 2 MARKET_BY_ORDER messages) integer
marketByPrice-KeyField	Secondary key for Level 2 messages	(required for Level 2 MARKET_BY_PRICE messages) integer
marketMakerKey-Field	Secondary key for Level 2 messages	(required for MARKET_MAKER messages) integer
nullField	A null value	(optional) A placeholder
respTypeNumField	Identifies type of message	(optional) integer
sequenceNumber	A unique number, assigned sequentially by the adapter to each incoming event whether it causes an update or not.	(optional)long

Field	Description	Datatype
<code>serviceName</code>	The name of the service from which RMDS received the market data from this RIC.	(optional) <code>string</code>
<code>updateNumber</code>	A unique number, assigned sequentially by the adapter to each incoming update.	(optional) <code>long</code>

Getting Stream Information from the Project

Gather the necessary information about the Reuters stream.

The first step in configuring the input adapter is to determine the source streams on Event Stream Processor that will receive the RMDS Market Data. If the Event Stream Processor project does not already include one or more streams for this purpose, define a new stream (or streams) for use with the Reuters adapter.

After you have chosen (or defined) the streams that will receive data from the Reuters OMM adapter, collect information about that stream from your project file. The Event Stream Processor project file contains one or more stream definitions. Each stream definition specifies a data stream that is instantiated when Event Stream Processor is started. The stream definition comprises:

- A unique ID for the stream
- A database store and output file for the stream data
- A list of the columns used as the unique key value for each row in the data stream

Once you have decided which streams will carry the RMDS data provided by the Reuters adapter, get information from the stream definition in the project file. There is no standard for project file names. Two Event Stream Processor installations may have completely different stream definitions, but the definition of any stream includes the same basic set of components.

These instructions refer to the example project to show what components of the stream configuration you must identify to configure the Reuters OMM adapter.

1. Open the project to which the adapter provides data. The Reuters OMM adapter distribution includes an example project, `$ESP_RMDSOMM_HOME/examples/example.ccl`, that contains schema definitions for three streams.
2. Find the name of the source stream. The opening `SourceStream` tag specifies the name of the stream as the value of the `id` attribute. The first source stream in this example is named “`marketByOrderStream`.”

The stream used for subscription by the Reuters OMM adapter must always be a source stream.

CHAPTER 2: Adapters Supported by Event Stream Processor

3. Determine the key fields. Check each column entry between the opening and closing SourceStream tags to see if the key attribute is set to true. In this example, “marketByOrderStream” has one key field: symbol.
4. Carefully note the number and order of the column entries in the source stream definition. In the input adapter map file, list the same set of data in the same order.

Creating the Input Map File

Use the procedure in the sample adapter map files provided in the `examples` subdirectory to create your own adapter map file.

1. Select or create a directory for your adapter map file.
2. Copy the contents of the `$ESP_RMDSOMM_HOME/examples` directory to that directory.
3. Use a text editor to modify the example files as necessary for your installation.

Running the Input Adapter

Run the Reuters OMM input adapter once you have configured it.

Prerequisites

Configure an adapter.

Task

1. Ensure that **esp_server** is running and that the project has been loaded and started.
2. If the Event Stream Processor is running with RSA authentication, start the adapter using:

```
esp_rmdsomm -a in -f mapfile -p cluster_host:cluster_port/  
workspace/project \  
-k <private_rsa_key_file> -c username
```

3. If Event Stream Processor is running with Kerberos/LDAP authentication, start the adapter using:

```
esp_rmdsomm -a in -f mapfile -p cluster_host:cluster_port/  
workspace/project \  
-c username:password
```

4. If Event Stream Processor is running with no authentication, start the adapter using:

```
esp_rmdsomm -a in -e -f mapfile -p cluster_host:cluster_port/  
workspace/project \  
-c username:password
```

5. The adapter starts the subscription by first connecting to Event Stream Processor and then connecting to RMDS. Both connections must be operational for any data to flow.

If you plan to direct the adapter's log output to `stderr`, as shown here, you may want to redirect `stderr` to a log file (for example, append `>& myrmdsommlog &` to the command line shown above).

Testing the Adapter

If the adapter is not working as expected, you can perform a quick sanity check by executing the `esp_rmdsomm` command and verifying whether the adapter is sending Reuters market data to Event Stream Processor.

- Execute `esp_rmdsomm`:

```
esp_rmdsomm -v
```

This command returns the version information. Ensure that the Event Stream Processor to which you are connecting is compatible with your version of the adapter.

- There are three quick ways to verify that the Reuters OMM adapter is sending Reuters market data to Event Stream Processor:
 - Use the Studio or the `esp_subscribe` command to check the output of the stream configured to receive Reuters data.
 - Use the tail command on the redirected adapter log file (specified in the adapter map file) or the Reuters subscriber log (specified in the configuration file `rmdsomm.cfg`) for activity.
 - Run the `esp_rmdsomm` command with the `-d7` option to produce verbose output.

Multiple RICs

When configuring an input adapter, specify multiple RICs if you are interested in more than one symbol.

There are several ways to do this:

- Specify each individual RIC by entering the name directly into the map file or use an XML ENTITY include file.
- Create a dynamic watch list, which employs Event Stream Processor to specify the list of RICs.
- Use a combination of the options above.

Individual RICs

Enter an item element declaration for each RIC you want in the `itemList` section of the map file.

Here is an example of this:

```
<itemLists service="SSL_PUB" stream="marketByOrderStream">
<itemList>
<item name="CSCO.O" />
<item name="K.N" />
<item name="KBN.N" />
<item name="KBR.N" />
<item name="ACAM.ARC" />
<item name="IBM.ARC" />
</itemList>
</itemLists>
```

It can become difficult to create and maintain your list of RICs this way if it is very large or changes frequently, for example, if you are attempting to track all of the stocks traded on the NYSE. All RICs for the same stream must use the same FID set. Since FIDs often vary by venue, use a different itemList and streamMap for each venue.

Creating a Dynamic Watch List

Creating a dynamic watch list is a bit more complex than creating individual RICs, but is also more flexible. You can specify a custom list of RICs.

Prerequisites

Define source stream (named MyInfoStream) to receive the data, and manually edit the list of RICs.

Task

This method is also dynamic: when inserts or deletes occur on the stream configured using these steps, RMDS subscriptions to the appropriate RICs are started or stopped.

1. Define a stream on Event Stream Processor (for example, MyListStream) which publishes to the adapter the list of RICs to which you wish to subscribe. This stream requires these columns:

Column	Description
symbol	Specifies an RIC symbol ticker (for example, CSCO.O) to which the adapter should subscribe.
service	Specifies the RMDS service on which to subscribe to obtain data for that RIC.
stream	Specifies the name of the stream (for example, MyInfoStream) on which the adapter publishes data for this RIC.

The stream can also include an optional fourth column, rfaQueue.

2. Define a second stream on Event Stream Processor (for example, MyInfoStream) that receives the data requested by the first stream.
3. Edit the map file to include the subscription.

```
<subscriptions>
<subscription name="subscription1" flags="BASE" >
<stream name="MyListStream" >
<name column="3" /> <!-- symbol -->
<field column="1" name="service"/>
<field column="2" name="stream"/>
</stream>
</subscription>
</subscriptions>
```

4. Specify the set of RICs you want and send them to the first stream you created (for example, `MyListStream`) to subscribe to them.
- a) Create a file with the same six columns that the stream expects in comma-separated values (CSV) format. The columns are: stream from which you are receiving data, opcode (the `p` in the example is for UPSERT), service, symbol, and stream to which you are sending data.

For example, use a text editor to open a new file (`RIClist.csv`) and put in these lines:

```
MyListStream,p,,IDN_RDF,MyInfoStream,CSCO.O
MyListStream,p,,IDN_RDF,MyInfoStream,K.N
MyListStream,p,,IDN_RDF,MyInfoStream,KBN.N
MyListStream,p,,IDN_RDF,MyInfoStream,KBN.R
MyListStream,p,,IDN_RDF,MyInfoStream,ACAM.ARC
MyListStream,p,,IDN_RDF,MyInfoStream,IBM.ARC
```

- b) Send the data from the file to Event Stream Processor using the `esp_convert` and `esp_upload` commands. The following example assumes that you have installed all Sybase command line tools in the default directories and added those directories to your `PATH` variable. If you have not, prepend the appropriate path to each command shown in this example.

For example, to send the file created in the previous step to Event Stream Processor running on port 11180 of your local server, enter:

```
cat RIClist.csv | esp_convert -c user:password -d "," \
-p localhost:11180/ws1/p1 | esp_upload -c user:password -p
localhost:11180/ws1/p1
```

- c) Start the adapter:

```
esp_rmdsomm -f mapfile -d7 -c user:password \
-p localhost:11180/ws1/p1 >& logfile &
```

If the adapter and Event Stream Processor are on different machines, enter the name of the remote host rather than `localhost` after the `-p` in the previous command.

Performance Tuning

There are several attributes you can use to fine-tune performance in an input adapter.

Attribute	Description
<code>flushInterval</code>	Specify an interval of time in microseconds (for example, 5000 microseconds = 5 milliseconds) to wait while accumulating data. At the end of this interval, any accumulated events are sent to Event Stream Processor. Send events less often to allow more events to be placed into a message resulting in a communications overhead savings. Use a nonzero <code>flushInterval</code> to make even accumulation time-based.

Attribute	Description
maxRecordsPerBlock	Specify the maximum number of accumulated events that the adapter should send to Event Stream Processor at a time. When the number of accumulated events is larger than this value, the envelope or transaction is broken into fragments that are less than or equal to the specified value. For example, if accumulated event counts of more than 1024 (which would immediately fill the Event Stream Processor Gateway's inbound queue) are expected, set maxRecordsPerBlock to a value like 500 to prevent the inbound queue from filling.
pendingLimit	Specify a threshold for the number of events that must accumulate before they are sent to Event Stream Processor. Set this parameter to zero to publish each event immediately when it happens (providing the lowest latency), at the expense of high network overhead (a TCP/IP packet for each update). If you set this parameter to a larger value, the adapter waits until that number of events have accumulated, packs them efficiently in TCP/IP packets, and sends them to Event Stream Processor. This saves communication work but increases latency on both the adapter and Event Stream Processor.
sendAsTransactions	<p>This parameter controls whether events are sent as an envelope or a transaction. You can specify this parameter on a per-stream basis.</p> <p>Set this parameter to true for Event Stream Processor to treat a group of updates as a single transaction. Transactions typically cause application-level workload savings, since Event Stream Processor collapses multiple updates to the same value in a transaction to a single update. If a transaction contains a delete, additional savings are achieved since updates prior to the delete can be discarded.</p> <p>If you set this parameter to false and you are not in low-latency mode (pendingLimit and flushInterval both set to zero), then use the maxRecordsPerBlock to control the size of the envelope. You still gain the communications overhead savings mentioned above, but not the transactional savings. This is the preferred configuration for applications that require every event to be sent separately, such as a market data compliance application.</p> <p>As a general rule, for quote-based applications, where only the most recent update matters, use transactions to be most efficient. For trades, however, where every event must be processed separately to compute a total volume, use envelopes instead.</p>

When you use both `flushInterval` and `pendingLimit`, no event waits longer than the time indicated in the `flushInterval` before being sent, and as long as the number of events specified in `pendingLimit` arrive, they are sent immediately. The adapter waits for the amount of time specified in the `flushInterval` and, if any events have accumulated, it sends them. If the number of `pendingLimit` events, or more accumulate while the adapter is sending the earlier events, the new events are sent immediately (without waiting for the `flushInterval`). If fewer than the number of `pendingLimit` events accumulate while the adapter is sending events, it waits for the `flushInterval` to elapse.

You can also use the `rfaQueue` attribute at the `itemLists`, `itemList`, or `item` element level. When specified, the `rfaQueue` attribute causes the element to be subscribed from Reuters on a named `rfaQueue`. Each `rfaQueue` is processed by its own thread within the Reuters adapter. Spreading requests across multiple threads can reduce latency and improve overall adapter throughput at the cost of greater CPU usage.

Since all images and updates come from Reuters on the same queue, the integrity of the order of arrival is maintained for any individual RIC. If you do not specify an `rfaQueue` for any of the elements, a single default queue (named "defaultQueue") is used for all RICs.

Output Adapter Configuration

Configure an output adapter to push data from Event Stream Processor to RMDS.

Before configuring an output adapter, decide which data to provide and how you want to set up your system.

You need to know the following about the Event Stream Processor instance from which you receive data.

- Possible security options in a cluster environment, and the workspace and project name.
- What type of authentication mechanism (Kerberos, RSA, LDAP, or none) does it use?

Data Decisions

Identify which columns from which streams in Event Stream Processor to publish data from.

The Reuters OMM adapter can rearrange the columns from a stream in any order. Its output can also include constants, and the published output can include values from more than one stream.

Consider these items when planning the output of the Reuters OMM Output adapter:

- For each stream for which to publish data, you must specify a unique key in the output adapter map file. Since this adapter sends data to RMDS, the unique identifier should be an RIC. For Market Price data the key can be just the RIC. For Level 2 data, the key must contain additional fields: `MarketbyPrice` requires `PRICE` and `SIDE`, and `MarketbyOrder` requires `ORDER_ID`, in addition to the RIC.
- Each data column you want to publish from any stream must map to a unique FID.
- Data from one column can be repeated in the published output, giving you a way to publish a `DateTime` value as separate `Date` and `Time` values.

CHAPTER 2: Adapters Supported by Event Stream Processor

- If the stream you are working with receives data about the same FID from more than one service, you can configure the adapter to differentiate these data items by service and transmit each service's data separately.
- The first time the Reuters OMM adapter publishes to RMDS, it publishes values for all the columns for which it is configured. After that initial image, the adapter only publishes updates for individual columns as these updates occur.

The datatype of the Event Stream Processor column must be compatible with the Reuters FID datatype that feeds it. This table shows possible matches between Event Stream Processor and FID datatypes:

Event Stream Processor Datatype	Reuters Datatype
integer	enumeration,time_seconds,uint32, uint64, or real32
long	int64 or uint64
money, float	real32 or real64
string	ASCII_string, RMTES_string
date, timestamp	date, time

Administrative Decisions

You have several administrative decisions to make in regards to the project.

Decision	Description
Session Name	An arbitrary string used to link the project and the adapter map file. Use it consistently.
Directories for logging and stream output	The adapter writes its own log messages and can generate a separate set of Reuters log messages. In the configuration, specify if and where these log files should be written.
Sybase user account	Specify a valid Event Stream Processor user account for the adapter to use, unless you specified no authentication when you started the Event Stream Processor.

Reuters Information

You need several pieces of information from Reuters to enable the Reuters OMM adapter to publish to the RMDS.

- The name of the Reuters service on which the adapter transmits data
- Up-to-date lists of valid Reuters Instrument Codes (RICs) and Field Identifiers (FID) used by RMDS

- The Product Permission Code assigned by Reuters

The adapter does not work with the Reuters Data Access Control System (DACS), so the Product Permission Code is needed to allow access to the information you are transmitting on the RMDS.

A list of FIDs, `$ESP_RMDSOMM_HOME/config/RDMFieldDictionary`, has been supplied as part of the Reuters adapter distribution. You can obtain the latest list and other information from your Reuters technical contact.

Getting Stream Information from the Project

Gather the necessary information from the project.

The first step in configuring the output adapter is determining which data elements from which streams on the Event Stream Processor are to be published. After you have chosen (or defined) a project containing the items for publication over RMDS via the Reuters adapter, collect information from the streams from which to obtain the data to send to RMDS.

Each stream definition specifies a data stream that is instantiated when Event Stream Processor is started up. The stream definition:

- Specifies a unique ID for the stream
- Identifies the columns used as the unique key value for each row in the data stream

Once you have decided which streams will provide the information to be sent to RMDS by the Reuters adapter, get information from the stream definition in the project file. There is no standard for project file names. Two Event Stream Processor installations may have completely different stream definitions, but the definition of any stream includes the same basic set of components.

1. Open the project from which the adapter is obtaining data. The Reuters OMM adapter distribution includes an example project in the `$ESP_RMDSOMM_HOME/examples/example.ccl` file.
2. From the definition of each stream defined in the project:
 - a) Obtain the name of the stream from the `id` attribute in the opening tag of that stream.
 - b) Verify that the `key` attribute is set to `true` for the column containing the RIC and note the column. In this example, the “`marketByOrderStream`” has the RIC in the column named “`symbol`,” which is identified as a key field.
 - c) Decide what data, if any, you want the adapter to send to RMDS.
3. Carefully note which streams contain data you want to send to RMDS, and where in the stream definition it is located.

In the output adapter map file, reference each of the columns you want to publish.

Creating the Output Map File

Create an adapter map file to configure the interface between the output adapter and Event Stream Processor.

There are sample adapter map files in the examples subdirectory.

1. Select or create a directory for your adapter map file.
2. Copy the contents of the `$ESP_RMDSOMM_HOME/examples` directory to that directory.
3. Use the text editor to modify the example files as necessary for your installation.

Running the Output Adapter

Run the adapter once you have configured it.

Prerequisites

Configure an adapter.

Task

1. Ensure that **esp_server** is running and that the project has been loaded and started.
2. Start the adapter:

```
esp_rmdsomm -a out -f mapfile -p cluster_host:cluster_port/  
workspace/project
```

The exact usage of the command depends on how you started your Event Stream Processor. You must invoke the adapter with compatible options. The command string shown invokes neither encryption nor authentication: you can specify either or both.

Note: If you plan to direct the adapter's log output to stderr, as shown here, you may want to redirect stderr to a log file (for example, append **>& myrmdsommlog &** to the command line shown above).

Testing the Adapter

If the adapter is not working as expected, you can perform a quick sanity check by executing the **esp_rmdsomm** command and verifying whether the adapter is sending Reuters market data to Event Stream Processor.

- Execute **esp_rmdsomm**:

```
esp_rmdsomm -v
```
- This command returns the adapter release number and the revision number of the source tree separated by an underscore character. Ensure that your version of the adapter is compatible with your version of Event Stream Processor.
- There are several ways to verify that the Reuters OMM adapter is publishing to RMDS:

- Use the **tail** command on the adapter log file to which console output was redirected or any of the Reuters publisher log files (specified in `rmdsomm.cfg`) to look for activity.
- Use the **esp_subscribe** command to look at the outbound stream and verify that values are changing.
- Use RMDS tools to subscribe to RICs provided by the output adapter.
- Use an input adapter to subscribe to the output adapter via the RMDS Market Data Hub (MDH).

Performance Tuning

You can improve the performance of output adapters by using multiple threads.

The subscriptions section of the output adapter map file can contain more than one subscription. Each subscription is instantiated on a separate thread so you can specify multiple subscription sections to gain the performance advantage of running on multiple threads.

Split Adapter Map Files

It can be advantageous to put part of your input or output adapter map file in a separate file.

For example, you might want to keep a subscription configuration in a map file, but break out the list of RICs you want the adapter to subscribe to.

The sample files in `$ESP_RMDSOMM_HOME/examples` demonstrate how this facilitates reuse. The `pubexample.omm.map.xml` map file references three “map fragment” files: `mbo.s.mf.xml`, `mbp.s.mf.xml`, and `mp.s.mf.xml` file. The `mbo.s.mf.xml` is also referenced by three other map files.

Map file fragments are reusable blocks of XML for constructing adapter map files, using the XML entity mechanism. File names are of the form `description.parent_element.mf.xml`. Some current descriptions are:

- **mbo** – MarketByOrder
- **mbp** – MarketByPrice
- **mp** – MarketPrice

And current parent_elements are:

- **sd** – Event Stream Processor model stream definition
- **sms** – Subscriber's streamMaps section
- **rfa** – common config section
- **sm** – Subscriber's streamMap
- **il** – Subscriber's itemList
- **s** – Publisher's stream

Therefore, it is evident that the `mbo.sm.mf.xml` file is a subscriber map fragment containing streamMap elements for a MARKET_BY_ORDER message.

Creating a Subordinate Map File

Create a subordinate map file to hold part of the map file configuration.

1. Go to the directory that contains the map File.
2. Create a new file with the extension `.xml`.
It is not necessary to add a declaration of the XML version.
3. Insert the selected content from the map file into the new file.
The content you add depends on which part of the map file you have decided to store separately.
4. (Optional) Add a comment to the new file.
5. Save the file when you are done.

Modifying the Main Map File

Modify the main map file to reference the subordinate file.

1. Make sure the first line of the main map file is:

```
<?xml version="1.0"?>
```
2. Between the XML version declaration and the opening adapter tag, add these lines:

```
<!DOCTYPE adapter SYSTEM "adapter.dtd" [  
]>
```
3. For each subordinate map file:
 - a) Between the two lines just added, add:

```
<!ENTITY SUBREF SYSTEM "SUBFILE">
```

where SUBREF is a string to reference the subordinate file and SUBFILE is the path and filename of the subordinate file itself. Enclose the path and filename in quotation marks.
 - b) Remove the content that you put in the subordinate map file.
 - c) Insert a string like the following to include the content from the subordinate map file:

```
&SUBREF;
```

where SUBREF is the string you specified to reference the subordinate file.

Command Usage

esp_ommsample

The **esp_ommsample** utility displays data received from the Reuters Market Data System (RMDS) to stdout.

Synopsis

```
esp_ommsample -u username [ OPTION ... ]
```

Description

The **esp_ommsample** utility operates as a data sink from RMDS for OMM messages. It enables you to see which fields are delivered and their values without setting up a Reuters OMM adapter and model.

esp_ommsample prints data to stdout, getting its configuration from the command line. You can run it for a specified period of time or stop it using Ctrl+C.

Required Arguments

- **-u username** – specify the user name with which to connect to RMDS [ENTER_VALID_USERNAME].

Options

- **-a FID_dictionary** – specify a dictionary to use instead of the default dictionary (`./config/RDMFieldDictionary`).
- **-A applicationId** – specify an ApplicationId to override the default (256).
- **-c Reuters config file** – specify the path and file name of the Reuters configuration file. Since this file is usually shared with the Reuters OMM adapter, this is, by default, set to `./config/rmdsomm.cfg`.
- **-e enum_defs** – specify a file name to override the default (`./config/enumtype.def`).
- **-f format** – specify the format (0, 1, 2, or 3) for update messages. The default, 0, is a multiline format with each value on a separate line. Specify 1 to get all of the values on one line, for example:

```
207 TRIN.O|TRDPRC_1=1.14|BID=1.13|ASK=1.17|ACVOL_1=1000|
ASK_TIME=10:26:2|
```

The RIC (TRIN.O) is prefaced by a millisecond timestamp and followed by FID=value pairs, delimited by "|". Specify 2 to have the FID numbers included along with the field names: field[FID]=value. Specify 3 for the tersest format: FID=value.

You can use separator characters for environment variables.

ESP_OMMSAMPLE_PAIR_SEPARATOR defaults to '='.

ESP_OMMSAMPLE_FIELD_SEPARATOR defaults to '|'.

ESP_OMMSAMPLE_TIMESTAMP_SEPARATOR defaults to ' '.

- **-h** – print this help message and exit.
- **-I instanceId** – specify an InstanceId to override the default (1).
- **-m type** – specify the message type (MMT) to use, where type is one of:

```
l = MarketPrice
m = MarketMaker
o = MarketByOrder
[ p = MarketByPrice ]
s = SymbolList
```

CHAPTER 2: Adapters Supported by Event Stream Processor

- **-p period** – specify the length of time (in seconds) to listen to updates before terminating. The default is 120.
- **-P position** – Specify a position to override the default (yourIP/net).
- **-r service** – specify the RMDS service: one that is a valid service name for your site. This value defaults to DF_EAP_LAB1, which is a service available on Reuters test lab.
- **-s symbol [symbol ...]** – specify one or more symbols (RICs) to which to subscribe. A space-separated list likely needs quotes to protect it from the shell.
- **-S file** – specify a file containing symbols (RICs) to which to subscribe. These are added to any RICs that have been specified via the -s option.
- **-v** – Show the version number and exit.

Examples

```
cd $ESP_RMDSOMM_HOME/bin
./esp_ommsample -u myUsername -r MY_SERVICE -m 1 -s GOOG.O >&
esp_ommsample.out &
```

esp_rmdsomm

The Reuters OMM Adapter adapts data from the Reuters Market Data System (RMDS) to the Event Stream Processor and vice versa.

Synopsis

```
esp_rmdsomm -f mapFile -p host:port/workspace/project [ OPTION ...]
```

Description

The **esp_rmdsomm** command can start an adapter as either a data source or sink to or from the Event Stream Processor to or from Reuters Market Data System (RMDS). To both subscribe from and publish data to RMDS, you must run two separate RMDS OMM adapter instances.

The metadata describing the connection has several parts, including a map file, configuration file, and possibly a configuration stream resident on a running instance of the Event Stream Processor.

Reuters OMM has several domains. Currently, only MARKET_PRICE, MARKET_BY_PRICE, and MARKET_BY_ORDER are fully supported. MARKET_MAKER is supported only for inbound streams. See the Reuters documentation for more information, including what FIDs to expect on the message domains.

The process runs as a daemon, getting its configuration from a map file. It handles SIGHUP; so you can enter `kill -s SIGHUP pid` on Linux or `kill -s HUP pid` on Solaris (where pid is the process ID of the **esp_rmdsomm** daemon, which you can obtain using the **ps** command) to gracefully shut down the adapter. Using the KILL signal rather than the HUP signal may prevent a complete clean up of system resources.

There are three directories underneath the directory where the adapter is installed containing additional information: `doc`, `examples`, and `config`. The `doc` directory contains Reuters README files that describe various configuration options. The `examples`

directory contains several example map files that demonstrate many features. The `config` directory contains example RMDS configuration files. Minimally, you must modify the RMDS config file with your site's specific information. Typically, you must also modify the map file to match the Event Stream Processor.

Required Arguments

- **-f mapFile** – specify the map file containing the metadata required to map the market data to/from RMDS.
- **-p hostname:port/workspace/project** – specify the URI to connect to the server (cluster manager). For example, `-p localhost:19011/default/prj1` specifies a project called `prj1` in the default workspace of an ESP cluster server using port 19011 on your localhost.

Options

- **-a in|out|interactive** – specifies whether the RMDS OMM adapter instance is passing data in to the Event Stream Processor or receiving data passed out from it. Valid values are `in`, `out` and `interactive`. Since the default value is `in`, this option is typically omitted when subscribing to market data.

For backward compatibility, "subscribe" (`in`) and "publish" (`out`) are still allowed, but the options have been deprecated.

- **-c user[:password]** – if you are using an authentication method that requires credentials (such as Kerberos, PAM, or RSA), this option passes those authentication credentials to Event Stream Processor. If Event Stream Processor successfully authenticates with these credentials, the connection is maintained, otherwise Event Stream Processor immediately closes the connection.
- **-d debugLevel** – sets the debug level. The valid range is 0–7, with 0 being minimal and 7 being verbose. By default it is set to 4.
- **-e** – negotiates encrypted OpenSSL sockets for all communication with the Event Stream Processor, which must be started in encrypted mode when using this option.
- **-F configFile** – specifies the RMDS Configuration file, overriding the configuration file specified in the map file.
- **-g gatewayHost** – explicitly specifies the Event Stream Processor gateway host.
- **-G** – uses Kerberos authentication. This option is required when the Event Stream Processor is started with the `-V gssapi` option.
- **-h** – print a short help message describing the syntax of this command.
- **-k privateRsaKeyFile** – perform authentication using the RSA private key file mechanism instead of password authentication. The `privateRSAKeyFile` must specify the absolute path filename of the private RSA key file. With this option enabled, the user name must be specified with the `-c` option, but the password is not required. In addition, Event Stream Processor must be started with the `-k` option.

CHAPTER 2: Adapters Supported by Event Stream Processor

- **-l0|1|2|3** – specify the location to which log messages get sent. Use 0 for no log messages, 1 to send to stderr only (the default), 2 to send to syslog only, and 3 to send to both stderr and syslog.
- **-r resubscribeInterval** – specify how many seconds to wait (default is 300) between attempts to resubscribe to a RIC. (If a subscription to a RIC is marked CLOSED or CLOSEDRECOVER, you must resubscribe to that RIC for data to flow.) To disable resubscription attempts, specify 0 as the value. Periodically resubscribing can compensate for a temporary condition where the source is not ready for subscribers. Each unsuccessful resubscribe attempt generates a failure event which may result in a status update marking the item stale.
- **-s streamName** – specify the stream to be used when running in discovery mode. This option is used by the connector start mechanism and specifies the single stream for which mapped columns have been discovered.
- **-v** – print the version of the RMDS OMM adapter and exit.
- **-w retrySeconds** – specify the number of seconds to wait between retries when connecting to the Event Stream Processor. The default is 5. Specify 0 to try only once.
- **-x optName** – specify various extra settings; use `-x help` to see a list of possible values.
- **-z publishCount** – specify the number of (2x) values to pass to the Event Stream Processor before terminating. By default this is 0, which means never terminate.
- **-Z subscribeCount** – Specify the number of (2x) values to pass to RMDS before terminating. By default this is 0, which means never terminate.

Examples

To start a Reuters OMM input adapter on the machine where you enter the command, using port 1099 and running project proj1 in workspace work02 using the myMap.xml map file:

```
esp_rmddsomm -c user:passwd -f myMap.xml -p localhost:1099/work02/  
proj1 -d 7 &> omm.in.log &
```

To start a Reuters OMM outbound adapter on a host named loki, using port 2010 and running project proj3 in workspace work01 using the myMap.xml map file:

```
esp_rmddsomm -a out -c user:passwd -f myMap.xml -p loki:2010/work01/  
proj3 -d 7 &> omm.out.log &
```

Environment Variables

The Reuters OMM adapters use environment variables to specify behavior.

Environment Variable	Used By	Description
ESP_ACCUMULATOR_DELAY	Input	(Expert) Delay connection to the Event Stream Processor (seconds).
ESP_DISABLE_REPORT_ENCODING_NULL	Output	Stop warning about blank-to-null conversions (bool) [false].

Environment Variable	Used By	Description
ESP_FLUSH_INTERVAL	Input	Override the publication flushInterval (microseconds).
ESP_INTRASUBSCRIBE_DELAY	Input	Override the map attribute (milliseconds).
ESP_LOG_CONFIG_EVENTS	Both	Set log level (1–7; 2x) for config event processing [-1]
ESP_MAX_RECORDS_PER_BLOCK	Input	Override the publication maxRecordsPerBlock (count).
ESP_PENDING_LIMIT	Input	Override the publication pendingLimit.
ESP_RETRY_INTERVAL	Both	Override the publication retryInterval.
ESP_RMDSOMM_DISPATCH	Both	(Expert) Dispatch RFA every N milliseconds [10,000].
ESP_RMDSOMM_EVENT_TRACE	Both	(Expert) Enable RFA event tracing every N event (int).
ESP_RMDSOMM_HOME	Both	Specify the installation directory.
ESP_RMDSOMM_PUBLISH_DEBUG_LEVEL	Output	Set to 7 to see values [not in -opt].
ESP_RMDSOMM_PUBLISH_DEBUG_SYMBOLS	Output	Contains a space-delimited list of symbols that are used when default behavior is overridden. If this environment variable is not set, all symbols are used.
ESP_RMDSOMM_SUBSCRIBE_DEBUG_LEVEL	Input	Set to 7 to see values [not in -opt].
ESP_RMDSOMM_SUBSCRIBE_DEBUG_SYMBOLS	Input	Contains a space-delimited list of symbols that are used when default behavior is overridden. If this environment variable is not set, all symbols are used.
ESP_RMDSOMM_SUBSCRIBE_SYMBOL_FORMAT	Input	Specify symbol list format: 0 for multiline; 1 for single line.
ESP_SEND_AS_TRANSACTIONS	Input	Override the map attribute.
ESP_SHOW_FIELD_INFO	Input	Show FID, column, spColumn, and stream name [false].

Environment Variable	Used By	Description
ESP_SHOW_SP_EVENT_DATA	Output	Set log level (1–7) for events from the Event Stream Processor [-1].

Input Adapter Map File

Shows the structure of the map file for the Reuters OMM input adapter.

```

adapter (required, limit one)
|----publication (required, limit one)
|----streamMaps (required, limit one)
|   '----streamMap (required)
|       |----dataField (required)
|       |----hiResTimestampField (optional)
|       |----imageField (required for L2 data)
|       |----itemName (required, limit one)
|       |----itemStale (optional)
|       |----marketByOrderKeyField (required)
|       |----marketByPriceKeyField (required)
|       |----marketMakerKeyField (required)
|       |----nullField (optional)
|       |----respTypeEnumField (optional)
|       |----sequenceNumber (optional)
|       |----serviceName (optional)
|       |----updateNumber (optional)
|----rfa (required, limit one)
|----itemLists (required, limit one)
|   '----itemList (required)
|       |----item (optional)
    
```

adapter

The adapter element is the root element of the map file.

Summary

```

adapter (required, limit one)
|----publication (required, limit one)
|----streamMaps (required, limit one)
|   '----streamMap (required)
|       |----dataField (required)
|       |----hiResTimestampField (optional)
|       |----imageField (required for L2 data)
|       |----itemName (required, limit one)
|       |----itemStale (optional)
|       |----marketByOrderKeyField (required)
|       |----marketByPriceKeyField (required)
|       |----marketMakerKeyField (required)
|       |----nullField (optional)
|       |----respTypeEnumField (optional)
|       |----sequenceNumber (optional)
|       |----serviceName (optional)
|       |----updateNumber (optional)
    
```

CHAPTER 2: Adapters Supported by Event Stream Processor

```
|----rfa (required, limit one)
|----itemLists (required, limit one)
|   '----itemList (required)
|       '----item (optional)
```

Parent

None

Children

The following child elements are defined for adapter. All of these elements must be present, and in the order specified.

Name	Requirement
publication	Exactly one required
streamMaps	Exactly one required
rfa	Exactly one required
itemLists	Exactly one required

Attributes

Name	Description	Requirement
name	A string that uniquely identifies this adapter (included in log entries)	Optional

Notes

None

Example

See the examples given for each of the component elements of the map.

dataField

In the streamMap definition, the dataField element maps a Reuters Field ID (FID) to one column in a source stream.

Summary

```
adapter (required, limit one)
|----publication (required, limit one)
|----streamMaps (required, limit one)
|   '----streamMap (required)
|       |----dataField (required)
|       |----hiResTimestampField (optional)
|       |----imageField (required for L2 data)
|       |----itemName (required, limit one)
|       |----itemStale (optional)
|       |----marketByOrderKeyField (required)
|       |----marketByPriceKeyField (required)
```

CHAPTER 2: Adapters Supported by Event Stream Processor

	----	marketMakerKeyField	(required)	
	----	nullField	(optional)	
	----	respTypeNumField	(optional)	
	----	sequenceNumber	(optional)	
	----	serviceName	(optional)	
	----	updateNumber	(optional)	
----	rfa		(required, limit one)	
----	itemLists		(required, limit one)	
	----	itemList	(required)	
		----	item	(optional)

Parent

streamMap

Children

None

Attributes

Name	Description	Requirement
name	The Reuters FID that identifies the data item that appears in this column of the source stream	Required
key	True or false, depending on whether this column is part of the source stream's unique key	See Notes

Notes

Each element in the **streamMap** section of the input adapter map file must represent a column in the row definition of the target source stream. (The order of the **streamMap** elements must mirror the order of the columns in the source stream.) If the column in the source stream is a data item (Bid, Ask, and so on), the corresponding **streamMap** entry must be a **dataField** element for which the name attribute identifies a specific FID. Any time RMDS publishes an update tagged with that FID, the adapter sends it to Event Stream Processor source stream as a value in the corresponding column.

Use the key attribute to set the value to true. If this column is not part of the stream's key, you can omit the key attribute.

The adapter uses the Event Stream Processor schema.

Example

```
<streamMap name="marketByOrder">
  <itemName key="true"/>
  <!-- serviceName / -->
  <sequenceNumber />
  <itemStale/>
  <dataField name="BID"/>
  <dataField name="ASK"/>
  <dataField name="TRDPRC_1"/>
```

```
<dataField name="ACVOL_1"/>
  <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>
```

This example maps columns 4–8 of the marketByOrder stream to the Reuters FIDs BID, ASK, TRDPRC_1, and ACVOL_1.

dateTimeField

In a streamMap, the **dateTimeField** element maps a Reuters date or time FID (or one of each) to a date column, a timestamp column, or both, in an Event Stream Processor source stream.

Summary

```
adapter (required, limit one)
  ----publication (required, limit one)
  ----streamMaps (required, limit one)
    '----streamMap (required)
      ----dataField (required)
      ----hiResTimestampField (optional)
      ----imageField (required for L2 data)
      ----itemName (required, limit one)
      ----itemStale (optional)
      ----marketByOrderKeyField (required)
      ----marketByPriceKeyField (required)
      ----marketMakerKeyField (required)
      ----nullField (optional)
      ----respTypeNumField (optional)
      ----sequenceNumber (optional)
      ----serviceName (optional)
      ----updateNumber (optional)
  ----rfa (required, limit one)
  ----itemLists (required, limit one)
    '----itemList (required)
      ----item (optional)
```

Parent

streamMap

Children

None

Attributes

Name	Description	Requirement
dateName	The FID of the date value provided by RMDS	See Note
timeName	The FID of the time value provided by RMDS	See Note

Notes

The `dateTime` datatype, which combines both date and time, is the most commonly used datatype for date/time information in Event Stream Processor data streams. In most cases, however, the updates provided by RMDS and brought in to the Event Stream Processor by the Reuters OMM adapter use separate FIDs for date and time.

To address this discrepancy, the map file provides the `dateTimeField` element, which provides separate attributes for date and time, allowing you to map two FIDs (one for date, one for time) to the same column in the source stream definition.

If `dateTime` is used, it must be used alone. The `dateName` and `timeName` attributes can be used either separately or together. One of these three attributes must be used.

The value for each FID must match one listed in the FID list referenced in the Reuters-side configuration file (the FID list provided with the adapter is named `appendix_a`). This file is referenced in the `rmdsomm.cfg` configuration file.

Example

```
<streamMap name="marketByOrderStream">
  <itemName key="true"/>
  <!-- serviceName / -->
  <sequenceNumber />
  <itemStale />
  <dataField name="BID"/>
  <dataField name="ASK"/>
  <dataField name="TRDPRC_1"/>
  <dataField name="ACVOL_1"/>
  <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>
```

This example maps the `TIMACT` and `ACTIV_DATE` FIDs together to the ninth column of the Event Stream Processor source stream `marketByOrderStream`.

hiResTimestampField

The `hiResTimestampField` element substitutes a high-resolution timestamp for the regular timestamp.

Summary

<code>adapter</code>	(required, limit one)
<code>----publication</code>	(required, limit one)
<code>----streamMaps</code>	(required, limit one)
<code>'----streamMap</code>	(required)
<code>----dataField</code>	(required)
<code>----hiResTimestampField</code>	(optional)
<code>----imageField</code>	(required for L2 data)
<code>----itemName</code>	(required, limit one)
<code>----itemStale</code>	(optional)
<code>----marketByOrderKeyField</code>	(required)
<code>----marketByPriceKeyField</code>	(required)

	----marketMakerKeyField	(required)
	----nullField	(optional)
	----respTypeNumField	(optional)
	----sequenceNumber	(optional)
	----serviceName	(optional)
	----updateNumber	(optional)
----rfa		(required, limit one)
'----itemLists		(required, limit one)
'----itemList		(required)
'----item		(optional)

Parent

streamMap

Children

None

Attributes

Name	Description	Requirement
name	long relative timestamp	required

Notes

This element can be used only on Solaris machines.

Example

```

<streamMap name="marketByOrderStream">
  <itemName key="true"/>
  <!-- serviceName / -->
  <hiResTimestampField name="TIME" />
  <itemStale/>
  <dataField name="BID" />
  <dataField name="ASK" />
  <dataField name="TRDPRC_1" />
  <dataField name="ACVOL_1" />
  <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE" />
</streamMap>

```

imageField

The **imageField** element indicates whether or not the Event Stream Processor row is part of a snapshot initial image.

Summary

adapter	(required, limit one)
----publication	(required, limit one)
----streamMaps	(required, limit one)
'----streamMap	(required)
----dataField	(required)
----hiResTimestampField	(optional)

CHAPTER 2: Adapters Supported by Event Stream Processor

```

|      |      | ----imageField          (required for L2 data)
|      |      | ----itemName           (required, limit one)
|      |      | ----itemStale         (optional)
|      |      | ----marketByOrderKeyField (required)
|      |      | ----marketByPriceKeyField (required)
|      |      | ----marketMakerKeyField (required)
|      |      | ----nullField       (optional)
|      |      | ----respTypeNumField (optional)
|      |      | ----sequenceNumber  (optional)
|      |      | ----serviceName   (optional)
|      |      | ----updateNumber   (optional)
| ----rfa
| ----itemLists
|     '----itemList
|         '----item          (optional)

```

Parent

streamMap

Children

None

Attributes

Name	Description	Requirement
name	Integer (1 if part of image, 0 if not)	Required for Level 2

Notes

The project should treat all rows of a snapshot image as one transaction.

Example

```

<name column="0" />
<keyField column="1" name="mbpkey" />
<imageField column="2" />
<!-- summary fields -->
<stale column="4" />
<field column="5" name="CURRENCY" />
<field column="6" name="ACTIV_DATE" />
<field column="7" name="PROD_PERM" />
<!-- end summary fields -->

```

item

The item element identifies an RIC to which the Reuters OMM adapter subscribes.

Summary

```

adapter          (required, limit one)
| ----publication (required, limit one)
| ----streamMaps (required, limit one)
|     '----streamMap (required)

```

```

|-----dataField (required)
|-----hiResTimestampField (optional)
|-----imageField (required for L2 data)
|-----itemName (required, limit one)
|-----itemStale (optional)
|-----marketByOrderKeyField (required)
|-----marketByPriceKeyField (required)
|-----marketMakerKeyField (required)
|-----nullField (optional)
|-----respTypeNumField (optional)
|-----sequenceNumber (optional)
|-----serviceName (optional)
|-----updateNumber (optional)
|-----rfa (required, limit one)
|-----itemLists (required, limit one)
|   '-----itemList (required)
|     '-----item (optional)

```

Parent

itemList

Children

None

Attributes

Name	Description	Requirement
name	An RIC to which the adapter will subscribe	Required
rfaQueue	A name for the rfaQueue, which, if provided, replaces the default rfaQueue name and cause a separate thread to be used for this queue	Optional
service	The name of a Reuters Service that provides incoming data through RMDS	Optional if already specified in the parent itemList or itemLists element, otherwise required
stream	The source stream on which updates for this RIC are brought to the Event Stream Processor	Optional if already specified in the parent itemList or itemLists element, otherwise required

Notes

The value for the name attribute should be a valid RIC on the service.

If you specify a stream name here, updates for this RIC are brought in to the Event Stream Processor on that stream. If you do not specify a stream here, the stream specified at the **itemList** level is used.

The stream you specify must match a **streamMap** defined elsewhere in the map file.

Example

```
<itemLists service="SSL_PUB" stream="marketByOrderStream">
  <itemList service="IDN_RDF" >
    <item name="EUR=" />
    <item name="EURJPY=" stream="stream6" />
  </itemList>
</itemLists>
```

These two **item** elements subscribe the adapter to the RICs EUR= and EURJPY=. The EUR= updates are sent to the stream marketByOrderStream which was set in the **itemLists** element. The EURJPY= updates are sent to the stream stream6, since the **item** level stream attribute overrides the **itemLists** level attribute.

itemList

The **itemList** element contains one or more instances of the **item** element.

Summary

```
adapter (required, limit one)
  ----publication (required, limit one)
  ----streamMaps (required, limit one)
    '----streamMap (required)
      ----dataField (required)
      ----hiResTimestampField (optional)
      ----imageField (required for L2 data)
      ----itemName (required, limit one)
      ----itemStale (optional)
      ----marketByOrderKeyField (required)
      ----marketByPriceKeyField (required)
      ----marketMakerKeyField (required)
      ----nullField (optional)
      ----respTypeNumField (optional)
      ----sequenceNumber (optional)
      ----serviceName (optional)
      ----updateNumber (optional)
  ----rfa (required, limit one)
  ----itemLists (required, limit one)
    '----itemList (required)
      '----item (optional)
```

Parent

itemLists

Children

Name	Requirement
item	zero or more required

Attributes

Name	Description	Requirement
rfaQueue	A name for the rfaQueue, which if provided, it replaces the default rfaQueue name and causes a separate thread to be used for this queue	optional
service	The name of a Reuters Service that provides incoming data through RMDS	optional if already specified in the parent itemLists element or in all child item elements, otherwise required
stream	The name of an Event Stream Processor source stream that will receive updates on the RICs specified in this list of items	optional if already specified in the parent itemLists element or in all child item elements, otherwise required

Notes

Configure the adapter to push updates for every item in this section to that stream (although you can override this specification at the item level) by specifying a stream name for this element.

The adapter supports more than one itemList element under itemLists; this allows you to configure one instance of the adapter to direct updates from two or more groups of RICs to different Event Stream Processor source streams.

The stream you specify must match one of the streamMaps defined elsewhere in the map file (by the value of the streamMap's name attribute).

Use the rfaQueue attribute to control scalability.

Example

```
<itemLists service="SSL_PUB" stream="marketByOrderStream">
  <itemList service="IDN_RDF" >
    <item name="EUR=" />
    <item name="EURJPY=" stream="stream6" />
  </itemList>
</itemLists>
```

This **itemList** element sets the service attribute to IDN_RDF, overriding the SSL_PUB service attribute defined in the parent **itemLists** element.

itemLists

The **itemLists** element contains one or more instances of the **itemList** element.

Summary

```

adapter (required, limit one)
  ----publication (required, limit one)
  ----streamMaps (required, limit one)
    '----streamMap (required)
      ----dataField (required)
      ----hiResTimestampField (optional)
      ----imageField (required for L2 data)
      ----itemName (required, limit one)
      ----itemStale (optional)
      ----marketByOrderKeyField (required)
      ----marketByPriceKeyField (required)
      ----marketMakerKeyField (required)
      ----nullField (optional)
      ----respTypeNumField (optional)
      ----sequenceNumber (optional)
      ----serviceName (optional)
      ----updateNumber (optional)
  ----rfa (required, limit one)
  ----itemLists (required, limit one)
    '----itemList (required)
      ----item (optional)
    
```

Parent

adapter

Children

Name	Requirement
itemList	One required, two or more supported

Attributes

Name	Description	Requirement
name	A string that appears in any adapter-related log entries	Optional
rfaQueue	A name for the rfaQueue, which if provided, it replaces the default rfaQueue name and causes a separate thread to be used for this queue (a default that you can override at the item level)	Optional

Name	Description	Requirement
service	The name of a Reuters Service that provides incoming data through RMDS (a default that you can override at the item level)	Optional if specified in the child itemLists or item elements or both, so that all child item elements either specify or inherit it, otherwise required
stream	The name of an Event Stream Processor source stream that receives updates on the RICs specified in the item lists in this section (a default that you can override at the item level)	Optional if specified in the child itemLists and/or item elements so that all child item elements either specify or inherit it, otherwise required

Notes

Each **itemList** instance in this section is a list of one or more RICs to which the adapter subscribes.

Get the value for service from your Reuters administrator.

Example

```
<itemLists service="SSL_PUB" stream="marketByOrderStream">
  <itemList service="IDN_RDF" >
    <item name="EUR=" />
    <item name="EURJPY=" stream="stream6" />
  </itemList>
</itemLists>
```

This **itemLists** element sets the service attribute to SSL_PUB and the stream attribute to marketByOrderStream. These attributes are either inherited, or overridden at the **itemList** and/or **item** level.

itemName

In the **streamMap** definition, the **itemName** element identifies the row in the Event Stream Processor source stream that carries the RIC from the RMDS update.

Summary

```
adapter (required, limit one)
  ----publication (required, limit one)
  ----streamMaps (required, limit one)
    '----streamMap (required)
      ----dataField (required)
      ----hiResTimestampField (optional)
      ----imageField (required for L2 data)
      ----itemName (required, limit one)
      ----itemStale (optional)
      ----marketByOrderKeyField (required)
      ----marketByPriceKeyField (required)
```

CHAPTER 2: Adapters Supported by Event Stream Processor

	----marketMakerKeyField	(required)
	----nullField	(optional)
	----respTypeNumField	(optional)
	----sequenceNumber	(optional)
	----serviceName	(optional)
	----updateNumber	(optional)
----rfa		(required, limit one)
'----itemLists		(required, limit one)
'----itemList		(required)
'----item		(optional)

Parent

streamMap

Children

None

Attributes

Name	Description	Requirement
key	True or false, depending on whether or not this column is part of the source stream's unique key	See first note

Notes

You need not use the **key** attribute; it is present for backward compatibility.

Insert the **itemName** element in the streamMap to correspond with the column in the source stream that carries the RIC or symbol. If this column is part of the source stream's key, set the **key** attribute to true.

This element is one of the "pseudofields" that specify data items that are not part of the data feed coming directly from RMDS.

Example

```
<streamMap name="marketByOrderStream">
  <itemName key="true"/>
  <!-- serviceName / -->
  <sequenceNumber />
  <itemStale/>
  <dataField name="BID"/>
  <dataField name="ASK"/>
  <dataField name="TRDPRC_1"/>
  <dataField name="ACVOL_1"/>
  <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>
```

The first column of the source stream is identified as the one that carries the RIC value of any update from the adapter. It is also identified as part of the stream's key.

itemStale

In the **streamMap** definition, the **itemStale** element identifies a column in the Event Stream Processor source stream that carries an indicator of whether or not incoming RMDS data has gone stale.

Summary

```

adapter (required, limit one)
  ----publication (required, limit one)
  ----streamMaps (required, limit one)
    '----streamMap (required)
      ----dataField (required)
      ----hiResTimestampField (optional)
      ----imageField (required for L2 data)
      ----itemName (required, limit one)
      ----itemStale (optional)
      ----marketByOrderKeyField (required)
      ----marketByPriceKeyField (required)
      ----marketMakerKeyField (required)
      ----nullField (optional)
      ----respTypeNumField (optional)
      ----sequenceNumber (optional)
      ----serviceName (optional)
      ----updateNumber (optional)
  ----rfa (required, limit one)
  ----itemLists (required, limit one)
    '----itemList (required)
      ----item (optional)

```

Parent

streamMap

Children

None

Attributes

Name	Description	Requirement
name	A string that appears in any adapter-related log entries	optional

Notes

Use this element in the **streamMap** if one of the columns in the source stream is a "stale" column.

RMDS itself does not supply a stale flag with regular market data, although it may pass along such a flag if it is provided by another service you are subscribing to via RMDS. If this element is used in the **streamMap**, the adapter sends a non-zero update value if it receives a stale flag

CHAPTER 2: Adapters Supported by Event Stream Processor

from RMDS, or stops receiving any data from RMDS. It uses three sets of bits to indicate stale reasons.

Adapter Status Bits	Description
0	Unknown = initial state
1	ConnectionInLoss
2	ConnectionOutLoss
3–7	Reserved

Data Status Bits	Description
8	Data suspect
9	Unspecified (initializing)
10–15	Reserved

Stream Status Bits	Description
16	Unspecified = initializing
17	NonStreaming = configured as snapshot only
18	ClosedRecover = stream is closed but can be retried
19	Closed = stream is closed and not coming back
20	Redirected = part of failing over state
21	Stale = for OMM
22–24	Reserved

Example

```
<streamMap name="marketByOrderStream">
  <itemName key="true"/>
  <!-- serviceName / -->
  <sequenceNumber />
  <itemStale/>
  <dataField name="BID"/>
  <dataField name="ASK"/>
  <dataField name="TRDPRC_1"/>
  <dataField name="ACVOL_1"/>
  <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>
```

The fourth column of the source stream is identified as the one that is updated if the adapter receives a "stale" notification or stops receiving data from RMDS.

marketByOrderKeyField

The **marketByOrderKeyField** element is a secondary key for messages of the MARKET_BY_ORDER domain.

Summary

adapter	(required, limit one)
----publication	(required, limit one)
----streamMaps	(required, limit one)
'----streamMap	(required)
----dataField	(required)
----hiResTimestampField	(optional)
----imageField	(required for L2 data)
----itemName	(required, limit one)
----itemStale	(optional)
----marketByOrderKeyField	(required)
----marketByPriceKeyField	(required)
----marketMakerKeyField	(required)
----nullField	(optional)
----respTypeNumField	(optional)
----sequenceNumber	(optional)
----serviceName	(optional)
----updateNumber	(optional)
----rfa	(required, limit one)
'----itemLists	(required, limit one)
'----itemList	(required)
'----item	(optional)

Parent

streamMap

Children

None

Attributes

Name	Description	Requirement
name	string	required for Level 2

Notes

Typically, the ORDER_ID FID is specified as the secondary key.

Example

```
<streamMaps>
  <streamMap name="MarketByOrderStream"
messageType="MARKET_BY_ORDER">
    &marketByOrder;
  </streamMap>
  <streamMap name="MarketByPriceStream"
```

CHAPTER 2: Adapters Supported by Event Stream Processor

```

messageType="MARKET_BY_PRICE">
  &marketByPrice;
</streamMap>
<streamMap name="MarketMakerStream" messageType="MARKET_MAKER">
  &marketMaker;
</streamMap>
</streamMaps>

```

marketByPriceKeyField

The marketByPriceKeyField element is a secondary key for messages of the MARKET_BY_PRICE domain.

Summary

```

adapter (required, limit one)
  ----publication (required, limit one)
  ----streamMaps (required, limit one)
    '----streamMap (required)
      ----dataField (required)
      ----hiResTimestampField (optional)
      ----imageField (required for L2 data)
      ----itemName (required, limit one)
      ----itemStale (optional)
      ----marketByOrderKeyField (required)
      ----marketByPriceKeyField (required)
      ----marketMakerKeyField (required)
      ----nullField (optional)
      ----respTypeNumField (optional)
      ----sequenceNumber (optional)
      ----serviceName (optional)
      '----updateNumber (optional)
  ----rfa (required, limit one)
  ----itemLists (required, limit one)
    '----itemList (required)
      '----item (optional)

```

Parent

streamMap

Children

None

Attributes

Name	Description	Requirement
name	PRICE + SIDE as a string	required for Level 2

Notes

This element is not meant to be parsed by the Event Stream Processor; it is used only as a secondary key to keep orderbook rows for the same RIC.

Example

```
<streamMaps>
  <streamMap name="MarketByOrderStream"
messageType="MARKET_BY_ORDER">
    &marketByOrder;
  </streamMap>
  <streamMap name="MarketByPriceStream"
messageType="MARKET_BY_PRICE">
    &marketByPrice;
  </streamMap>
  <streamMap name="MarketMakerStream" messageType="MARKET_MAKER">
    &marketMaker;
  </streamMap>
</streamMaps>
```

marketMakerKeyField

The marketMakerKeyField element is a secondary key for messages of the MARKET_MAKER domain.

Summary

```
adapter (required, limit one)
  |----publication (required, limit one)
  |----streamMaps (required, limit one)
  |   |----streamMap (required)
  |   |   |----dataField (required)
  |   |   |----hiResTimestampField (optional)
  |   |   |----imageField (required for L2 data)
  |   |   |----itemName (required, limit one)
  |   |   |----itemStale (optional)
  |   |   |----marketByOrderKeyField (required)
  |   |   |----marketByPriceKeyField (required)
  |   |   |----marketMakerKeyField (required)
  |   |   |----nullField (optional)
  |   |   |----respTypeNumField (optional)
  |   |   |----sequenceNumber (optional)
  |   |   |----serviceName (optional)
  |   |   |----updateNumber (optional)
  |   |----rfa (required, limit one)
  |   |----itemLists (required, limit one)
  |   |   |----itemList (required)
  |   |   |   |----item (optional)
```

Parent

streamMap

Children

None

Attributes

Name	Description	Requirement
name	string, typically MMID	required for Level 2

Notes

None

Example

```
<streamMaps>
  <streamMap name="MarketByOrderStream"
messageType="MARKET_BY_ORDER">
    &marketByOrder;
  </streamMap>
  <streamMap name="MarketByPriceStream"
messageType="MARKET_BY_PRICE">
    &marketByPrice;
  </streamMap>
  <streamMap name="MarketMakerStream" messageType="MARKET_MAKER">
    &marketMaker;
  </streamMap>
</streamMaps>
```

nullField

In a **streamMap**, the **nullField** element acts as a placeholder that always delivers a NULL value to the Event Stream Processor source stream. This lets you add extra fields to a source stream to get the configuration you want.

Summary

```
adapter (required, limit one)
  ----publication (required, limit one)
  ----streamMaps (required, limit one)
    '----streamMap (required)
      ----dataField (required)
      ----hiResTimestampField (optional)
      ----imageField (required for L2 data)
      ----itemName (required, limit one)
      ----itemStale (optional)
      ----marketByOrderKeyField (required)
      ----marketByPriceKeyField (required)
      ----marketMakerKeyField (required)
      ----nullField (optional)
      ----respTypeNumField (optional)
      ----sequenceNumber (optional)
      ----serviceName (optional)
```

'----updateNumber	(optional)
'----rfa	(required, limit one)
'----itemLists	(required, limit one)
'----itemList	(required)
'----item	(optional)

Parent

streamMap

Children

None

Attributes

Name	Description	Requirement
name	A string that appears in any adapter-related log entries	optional
dateName	A string that appears in any adapter-related log entries	optional
timeName	A string that appears in any adapter-related log entries	optional

Notes

When experimenting with a project, you can replace a **dataField** or **dateTimeField** element with a **nullField** to temporarily stop feeding data into any column of the stream.

You need not modify any attribute(s) of the **dataField** or **dateTimeField** you are temporarily replacing, as the following example shows.

Example

```
<streamMap name="marketByOrderStream">
  <itemName key="true"/>
  <!-- serviceName / -->
  <sequenceNumber />
  <itemStale/>
  <dataField name="BID"/>
  <dataField name="ASK"/>
  <nullField name="TRDPRC_1"/>
  <dataField name="ACVOL_1"/>
  <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>
```

The sixth column of the source stream is identified as a placeholder that receives a null value in each update from the adapter. It includes the name of the **dataField** that it replaces for debugging purposes.

CHAPTER 2: Adapters Supported by Event Stream Processor

publication

The **publication** element specifies basic publishing information for this instance of the adapter.

Summary

```
adapter (required, limit one)
  ----publication (required, limit one)
  ----streamMaps (required, limit one)
    '----streamMap (required)
      ----dataField (required)
      ----hiResTimestampField (optional)
      ----imageField (required for L2 data)
      ----itemName (required, limit one)
      ----itemStale (optional)
      ----marketByOrderKeyField (required)
      ----marketByPriceKeyField (required)
      ----marketMakerKeyField (required)
      ----nullField (optional)
      ----respTypeNumField (optional)
      ----sequenceNumber (optional)
      ----serviceName (optional)
      ----updateNumber (optional)
  ----rfa (required, limit one)
  ----itemLists (required, limit one)
    '----itemList (required)
      ----item (optional)
```

Parent

adapter

Children

None

Attributes

Name	Description	Requirement
flushInterval	Specify the number of microseconds the adapter allows events to accumulate before sending them to the Event Stream Processor. A nonzero flushInterval makes event accumulation time-based.	Optional (the default is 1000)
intraSubscribeDelay	Specify the number of milliseconds the adapter pauses between subscription requests.	Optional (the default is 100)

Name	Description	Requirement
maxRecordsPerBlock	Specify the maximum number of accumulated events that the adapter sends to the Event Stream Processor at a time. This reduces the size of each transaction or envelope fragment when there is a large number of accumulated events. For example, if 140 events have accumulated and maxRecordsPerBlock is set to 50, the adapter sends the envelope or transaction as three fragments.	Optional (the default is 256)
name	Specify a string that identifies the adapter instance in log file entries.	Optional
pendingLimit	Specify the number of events that may accumulate before the adapter sends them in to the Event Stream Processor. Using a pendingLimit makes the event accumulation count-based.	Optional (the default is 256)
retryInterval	Specify the number of seconds for which the adapter waits between attempts to connect to RMDS before shutting down.	Optional (the default is 5)
sendAsTransactions	Set to true to treat a group of updates as a single transaction or false to treat them as separate rows within an envelope.	Optional (the default is false)

Notes

You can optimize the adapter's performance using the pendingLimit and flushInterval attributes, along with the maxRecordsPerBlock and sendAsTransactions attributes from the Pub/Sub interface that the adapter uses to communicate with the Event Stream Processor. See *Performance Tuning* for details.

Some venues send initial images as multipart messages, which may produce large data sets. The intraSubscribeDelay attribute provides the ability to pace these subscriptions and prevents the adapter from being overwhelmed by initial images. The default value is zero, which is suitable for short RIC lists. When intraSubscribeDelay is set to a nonzero value, the adapter pauses between subscription requests for the specified number of milliseconds. The suggested value is ten (10).

Example

```
<publication name="RMDS Adapter - low latency" retryInterval="5"
  flushInterval="0" pendingLimit="0" sendAsTransactions="0" />
```

respTypeNumField

The **respTypeNumField** element populates a column with the RMDS respTypeNum value.

Summary

```

adapter (required, limit one)
  ----publication (required, limit one)
  ----streamMaps (required, limit one)
    '----streamMap (required)
      ----dataField (required)
      ----hiResTimestampField (optional)
      ----imageField (required for L2 data)
      ----itemName (required, limit one)
      ----itemStale (optional)
      ----marketByOrderKeyField (required)
      ----marketByPriceKeyField (required)
      ----marketMakerKeyField (required)
      ----nullField (optional)
      ----respTypeNumField (optional)
      ----sequenceNumber (optional)
      ----serviceName (optional)
      ----updateNumber (optional)
  ----rfa (required, limit one)
  ----itemLists (required, limit one)
    '----itemList (required)
      ----item (optional)
    
```

Parent

streamMap

Children

None

Attributes

Name	Description	Requirement
name	A string used in adapter-related log entries	optional

Notes

For an initial snapshot image, **respTypeNumField** has a value of 1 for UNSOLICITED or 0 for SOLICITED. Updates may have other values. See the RMDS documentation for more details.

Example

```

<itemName key="true" /> <!-- str: the RIC -->
<marketByPriceKeyField key="true"/> <!-- str: SIDE + PRICE as a key -->
<imageField name="imageIn" />
<updateNumber name="upd" /> <!-- generated by Adapter -->
<respTypeNumField name="rtn" />
    
```

rfa

The **rfa** element links the subscriber map file to the Reuters-side configuration file.

Summary

```

adapter (required, limit one)
  ----publication (required, limit one)
  ----streamMaps (required, limit one)
    '----streamMap (required)
      ----dataField (required)
      ----hiResTimestampField (optional)
      ----imageField (required for L2 data)
      ----itemName (required, limit one)
      ----itemStale (optional)
      ----marketByOrderKeyField (required)
      ----marketByPriceKeyField (required)
      ----marketMakerKeyField (required)
      ----nullField (optional)
      ----respTypeNumField (optional)
      ----sequenceNumber (optional)
      ----serviceName (optional)
      ----updateNumber (optional)
  ----rfa (required, limit one)
  ----itemLists (required, limit one)
    '----itemList (required)
      ----item (optional)
    
```

Parent

adapter

Children

None

Attributes

Name	Description	Requirement
config	The absolute path and file name of the Reuters-side configuration file for subscription (the sample file supplied with the adapter is at \$ESP_RMDSOMM_HOME/config/rmdsomm.cfg).	Required
configDatabaseName	Must be set to RFA.	Required
enumFile	The full path name of the Reuters-supplied file that lists each enumerated type along with the range of values it can take	See first Note

CHAPTER 2: Adapters Supported by Event Stream Processor

Name	Description	Requirement
fidFile	The full path name of the Reuters-supplied file that lists all of the valid FIDs	See second Note
sessionName	A reference to a session name defined in the Reuters-side configuration file for subscription	Required
blank	Specifies a marker to use for blanks	Optional
blankInt32	Specifies a marker to use for blank Int32 fields	Optional
blankInt64	Specifies a marker to use for blank Int64 fields	Optional
blankMoney	Specifies a marker to use for blank Money fields	Optional
blankString	Specifies a marker to use for blank String fields	Optional
blankDate	Specifies a marker to use for blank Date fields	Optional
blankTimestamp	Specifies a marker to use for blank Timestamp fields	Optional
blankDouble	Specifies a marker to use for blank Double fields	Optional

Notes

The default enumFile is `$ESP_RMDSOMM_HOME/config/enumtype.def`.

The default fidFile is `$ESP_RMDSOMM_HOME/config/RDMFieldDictionary`.

You can specify another file for either of these defaults.

Example

```
<rfa config="$ESP_RMDSOMM_HOME/config/rmdsomm.cfg"
    sessionName="Session1" />
```

This example points the Reuters OMM adapter to the Reuters-side configuration in the file `rmdsomm.cfg`. The list line in this configuration file is:

```
\Sessions\Session1\connectionList =
"Connection_SSLED"
```

This line defines a session name that is referenced by other lines in the configuration file. When the map file references a session name in the `sessionName` attribute, it links the adapter to the Reuters-side configuration parameters identified by that name.

sequenceNumber

In the **streamMap** definition, the **sequenceNumber** element maps a column in Event Stream Processor source stream that is populated by a unique number generated by the adapter, not provided as part of the data from RMDS.

Summary

```

adapter (required, limit one)
  ----publication (required, limit one)
  ----streamMaps (required, limit one)
    '----streamMap (required)
      ----dataField (required)
      ----hiResTimestampField (optional)
      ----imageField (required for L2 data)
      ----itemName (required, limit one)
      ----itemStale (optional)
      ----marketByOrderKeyField (required)
      ----marketByPriceKeyField (required)
      ----marketMakerKeyField (required)
      ----nullField (optional)
      ----respTypeNumField (optional)
      ----sequenceNumber (optional)
      ----serviceName (optional)
      ----updateNumber (optional)
  ----rfa (required, limit one)
  ----itemLists (required, limit one)
    '----itemList (required)
      ----item (optional)

```

Parent

streamMap

Children

None

Attributes

Name	Description	Requirement
key	true or false, depending on whether this column is part of the source stream's unique key	See Note
name	a string that appears in log entries	Optional

Notes

The adapter maintains a separate counter for each RIC to which it is subscribed. Each time it receives an update for an RIC, it increments its counter for that RIC. This number is the one sent to the source stream column mapped by the **sequenceNumber** element.

Source stream definitions include a column specification similar to:

CHAPTER 2: Adapters Supported by Event Stream Processor

```
<Column datatype="long" name="Id"/>
```

This line specifies a unique ID for the source stream. The **sequenceNumber** pseudo-field is a good match for this column in the input adapter map file.

You must use the key attribute to set the value to true. If this column is not part of the stream's key, you can omit this attribute.

Example

```
<streamMap name="marketByOrderStream">
  <itemName key="true"/>
  <!-- serviceName / -->
  <sequenceNumber />
  <itemStale/>
  <dataField name="BID"/>
  <dataField name="ASK"/>
  <dataField name="TRDPRC_1"/>
  <dataField name="ACVOL_1"/>
  <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>
```

The third column of the source stream is mapped to the sequence number provided by the adapter. This column is also identified as part of the source stream's unique key.

serviceName

In the **streamMap** definition, the **serviceName** element maps a column in the Event Stream Processor source stream to the service identifier that the adapter provides.

Summary

```
adapter (required, limit one)
  ----publication (required, limit one)
  ----streamMaps (required, limit one)
    '----streamMap (required)
      ----dataField (required)
      ----hiResTimestampField (optional)
      ----imageField (required for L2 data)
      ----itemName (required, limit one)
      ----itemStale (optional)
      ----marketByOrderKeyField (required)
      ----marketByPriceKeyField (required)
      ----marketMakerKeyField (required)
      ----nullField (optional)
      ----respTypeNumField (optional)
      ----sequenceNumber (optional)
      ----serviceName (optional)
      ----updateNumber (optional)
  ----rfa (required, limit one)
  ----itemLists (required, limit one)
    '----itemList (required)
      '----item (optional)
```

The identifier provided by **serviceName** can potentially be used to provide namespace scope for a RIC that was provided by two different services to which you subscribed.

Parent

streamMap

Children

None

Attributes

Name	Description	Requirement
key	true or false, depending on whether this column is part of the source stream's unique key	see Notes

Notes

You must use the key attribute to set the value to true. If this column is not part of the stream's key, you can omit this attribute.

Example

```
<streamMap name="marketByOrderStream">
  <itemName key="true"/>
  <!-- serviceName / -->
  <sequenceNumber />
  <itemStale/>
  <dataField name="BID"/>
  <dataField name="ASK"/>
  <dataField name="TRDPRC_1"/>
  <dataField name="ACVOL_1"/>
  <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>
```

In this example, no column of the source stream is mapped to the service name provided by the adapter because it is commented out.

streamMap

The **streamMap** element of the input map file defines the mappings between the columns of an Event Stream Processor source stream and the RMDS FIDs being subscribed to by the adapter.

Summary

```
adapter (required, limit one)
| ----publication (required, limit one)
| ----streamMaps (required, limit one)
| | ----streamMap (required)
| | | ----dataField (required)
| | | ----hiResTimestampField (optional)
| | | ----imageField (required for L2 data)
```

CHAPTER 2: Adapters Supported by Event Stream Processor

```

|-----itemName (required, limit one)
|-----itemStale (optional)
|-----marketByOrderKeyField (required)
|-----marketByPriceKeyField (required)
|-----marketMakerKeyField (required)
|-----nullField (optional)
|-----respTypeNumField (optional)
|-----sequenceNumber (optional)
|-----serviceName (optional)
|-----updateNumber (optional)
|-----rfa (required, limit one)
|-----itemLists (required, limit one)
|-----itemList (required)
|-----item (optional)

```

Parent

streamMaps

Children

The following child elements are defined for **streamMap**. These child elements can occur in any order, but for a specific **streamMap**, the order of the child elements must mirror the order of the columns of the source stream (as defined in the project). This is how the adapter is configured to deliver RMDS updates to the appropriate rows in the source stream.

Name	Requirement
dataField	One required, two or more supported
dateTimeField	Zero or more supported
imageField	Required for Level 2 data
itemName	One required, two or more supported
itemStale	Zero or one supported
marketByOrderKeyField	Required for Level 2 MARKET_BY_ORDER messages
marketByPriceKeyField	Required for Level 2 MARKET_BY_PRICE messages
marketMakerKeyField	Required for Level 2 MARKET_MAKER messages
nullField	Zero or more supported
respTypeNumField	Zero or more supported
sequenceNumber	Zero or more supported
serviceName	Zero or more supported

Name	Requirement
updateNumber	Zero or more supported

Attributes

Name	Description	Requirement
name	Identifies the source stream to which the RMDS updates are mapped; must match the name of a source stream defined in the Event Stream Processor project	Required
sendAsTransactions	True to treat a group of updates as a single transaction, or false to treat them as separate rows within an envelope	Optional (the default is false)

Notes

None

Example

```
<streamMaps>
  <streamMap name="marketByOrderStream">
    <itemName key="true"/>
    <!-- serviceName / -->
    <sequenceNumber />
    <itemStale/>
    <dataField name="BID"/>
    <dataField name="ASK"/>
    <dataField name="TRDPRC_1"/>
    <dataField name="ACVOL_1"/>
    <dateTimeField timeName="TIMACT"
dateName="ACTIV_DATE"/>
  </streamMap>
</streamMaps>
```

This example maps a set of the adapter's updates to an Event Stream Processor source stream named `marketByOrderStream`. All updates going to this source stream are added using the `upsert` opcode.

The RICs for which updates are sent to this source stream are specified in an `itemList` elsewhere in the map file that also references `marketByOrderStream`.

streamMaps

The **streamMaps** element of the input map file contains one or more **streamMap** elements.

Summary

```
adapter (required, limit one)
|----publication (required, limit one)
```

CHAPTER 2: Adapters Supported by Event Stream Processor

----streamMaps	(required, limit one)
'----streamMap	(required)
----dataField	(required)
----hiResTimestampField	(optional)
----imageField	(required for L2 data)
----itemName	(required, limit one)
----itemStale	(optional)
----marketByOrderKeyField	(required)
----marketByPriceKeyField	(required)
----marketMakerKeyField	(required)
----nullField	(optional)
----respTypeNumField	(optional)
----sequenceNumber	(optional)
----serviceName	(optional)
----updateNumber	(optional)
----rfa	(required, limit one)
----itemLists	(required, limit one)
'----itemList	(required)
----item	(optional)

Parent
adapter

Children

Name	Requirement
streamMap	One required, two or more supported

Attributes
None

Notes

Each **streamMap** instance in this section maps incoming FIDs from the Reuters adapter to columns in an Event Stream Processor source stream.

A stream must have a **streamMap**.

Example

```
<streamMaps>
  <streamMap name="marketByOrderStream">
    <itemName key="true"/>
    <!-- serviceName / -->
    <sequenceNumber />
    <itemStale/>
    <dataField name="BID"/>
    <dataField name="ASK"/>
    <dataField name="TRDPRC_1"/>
    <dataField name="ACVOL_1"/>
    <dateTimeField timeName="TIMACT"
dateName="ACTIV_DATE"/>
```

```

    </streamMap>
</streamMaps>

```

updateNumber

In the **streamMap** definition, the **updateNumber** element maps a column in Event Stream Processor source stream that are populated by a unique number generated by the adapter, not provided as part of the data from RMDS.

Summary

```

adapter                                     (required, limit one)
  |----publication                          (required, limit one)
  |----streamMaps                           (required, limit one)
  |   '----streamMap                        (required)
  |       |----dataField                     (required)
  |       |----hiResTimestampField          (optional)
  |       |----imageField                    (required for L2 data)
  |       |----itemName                      (required, limit one)
  |       |----itemStale                     (optional)
  |       |----marketByOrderKeyField        (required)
  |       |----marketByPriceKeyField       (required)
  |       |----marketMakerKeyField        (required)
  |       |----nullField                     (optional)
  |       |----respTypeNumField            (optional)
  |       |----sequenceNumber               (optional)
  |       |----serviceName                  (optional)
  |       |----updateNumber                 (optional)
  |----rfa                                  (required, limit one)
  |----itemLists                            (required, limit one)
  |   '----itemList                         (required)
  |       '----item                          (optional)

```

Parent

streamMap

Children

None

Attributes

Name	Description	Requirement
key	true or false, depending on whether this column is part of the source stream's unique key	See Notes
name	A string that appears in log entries	Optional

Notes

The adapter infers whether or not a column is part of the stream's unique key from the schema; the key attribute is included here only for backward compatibility.

CHAPTER 2: Adapters Supported by Event Stream Processor

The adapter maintains a separate counter for each RIC to which it is subscribed. Each time it receives an update for a RIC, it increments its counter for that RIC. This number is sent to the column in the stream mapped by the **updateNumber** element.

Many source stream definitions include a column specification similar to:

```
<Column datatype="int64" name="Id" />
```

This line specifies a unique ID for the source stream. The **updateNumber** pseudo-field is a good match for this column in the input adapter map file.

Example

```
<streamMap name="marketByOrderStream">
  <itemName key="true" />
  <updateNumber />
  <itemStale />
  <dataField name="BID" />
  <dataField name="ASK" />
  <dataField name="TRDPRC_1" />
  <dataField name="ACVOL_1" />
  <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE" />
</streamMap>
```

In this example, the second column of the source stream is mapped to the update number provided by the adapter. This column is also identified as part of the source stream's unique key. To see additional examples, look in the `$ESP_RMDSOMM_HOME/examples` directory.

Output Adapter Map File XML Syntax

The syntax of the map file for a Reuters OMM output adapter.

```
adapter (required, limit one)
|----rfa (required, limit one)
'----subscriptions (required, limit one)
  '----subscription (required)
    '----stream (required)
      |----name (required, limit one)
      |----stale (optional)
      |----field (required)
      '----constant (optional)
```

adapter

The **adapter** element is the root element of the map file.

Summary

```
adapter (required, limit one)
|----rfa (required, limit one)
'----subscriptions (required, limit one)
  '----subscription (required)
    '----stream (required)
      |----name (required, limit one)
      |----stale (optional)
```

```

|----field      (required)
'----constant  (optional)

```

Nest all configuration elements between the start and end **adapter** tags.

Parent

None

Children

The following child elements are defined for **adapter**. All of these elements must be present in the specified order.

Name	Requirement
rfa	Exactly one required
subscriptions	Exactly one required

Attributes

None

Notes

None

Example

See the examples for the child elements.

constant

The **constant** element defines a data item with a constant value that will be published to RMDS by the adapter.

Summary

```

adapter      (required, limit one)
|----rfa    (required, limit one)
'----subscriptions  (required, limit one)
    '----subscription  (required)
        '----stream    (required)
            |----name    (required, limit one)
            |----stale   (optional)
            |----field   (required)
            '----constant (optional)

```

Parent

stream

Children

None

Attributes

Name	Description	Requirement
name	The name associated with this data item in the image published by the adapter	Required
value	The value of this constant (always the same whenever this data item is published to RMDS)	Required

Notes

On start-up, the adapter publishes a complete image to RMDS, containing all data items defined in the map file. After that, the adapter publishes updated values for data items only when they change, unless Event Stream Processor goes stale and then recovers. This means that the value for **constant** is published only when a complete image is published.

Example

```
<stream name="stream1" >
  <name column="0" />
  <stale column="3" name="ACVOL_1" />
  <field column="1" name="DSPLY_NAME" />
  <field column="4" name="BID" precision="47" />
  <field column="5" name="ASK" precision="0" />
  <field column="6" name="TRDPRC_1" />
  <field column="7" name="ACVOL_1" />
  <constant name="PROD_PERM" value="1" />
</stream>
```

This example defines a constant called PROD_PERM, with the constant value 1, to be published with data values from stream1 under the publication name subscription1.

field

In a **stream** definition in an output adapter map file, **field** specifies a column in a stream to publish.

Summary

```
adapter (required, limit one)
  |----rfa (required, limit one)
  '----subscriptions (required, limit one)
    '----subscription (required)
      '----stream (required)
        |----name (required, limit one)
        |----stale (optional)
        |----field (required)
        '----constant (optional)
```

Parent
stream

Children

None

Attributes

Name	Description	Requirement
column	A number that represents the position of the source column in the stream being published from (the first column in the stream has the number 0)	Either column or columnName is required
columnName	The name of the column in the Event Stream Processor stream that carries the stream's unique identifier	Either column or columnName is required
name	The FID that identifies this data value when published to RMDS	Required
precision	An integer that specifies the total number of digits after the decimal point in the published value (for example, 1.23 has a precision of 2)	Optional

Notes

The precision attribute should be included only for columns of datatype double.

Example

```
<stream name="stream1" >
  <name column="0" />
  <stale column="3" name="ACVOL_1" />
  <field column="1" name="DSPLY_NAME" />
  <field column="4" name="BID" precision="47" />
  <field column="5" name="ASK" precision="0" />
  <field column="6" name="TRDPRC_1" />
  <field column="7" name="ACVOL_1" />
  <constant name="PROD_PERM" value="1" />
</stream>
```

The adapter is configured to publish updates from the fourth, fifth, sixth and seventh columns of the Event Stream Processor stream named stream1 as data items named BID, ASK, TRDPRC_1, and ACVOL_1, respectively.

name

In a **stream** definition in an output adapter map file, **name** specifies the column in the source stream that provides the value to use to identify each update.

Summary

```
adapter (required, limit one)
|----rfa (required, limit one)
```

```
'----subscriptions (required, limit one)
  '----subscription (required)
    '----stream (required)
      |----name (required, limit one)
      |----stale (optional)
      |----field (required)
      '----constant (optional)
```

Parent
stream

Children
None

Attributes

Name	Description	Requirement
column	A number that represents the position of the column in the stream that carries the stream's unique identifier (the first column in the stream is number 0)	Either column or columnName
columnName	The name of the column in the stream that carries the stream's unique identifier	Either column or columnName

Notes

The output adapter uses RMDS as a simple message bus; published updates need not conform to Reuters protocols. This means that the column specified by this element does not have to be a Reuters RIC, but it must follow Reuters RIC syntax.

If the source stream's unique key is a composition of two or more columns, you can use the name element in combination with one or more instances of the service element to configure the adapter to publish updates with completely unique names.

Example

```
<stream name="stream1" >
  <name column="0" />
  <stale column="3" name="ACVOL_1" />
  <field column="1" name="DSPLY_NAME" />
  <field column="4" name="BID" precision="47" />
  <field column="5" name="ASK" precision="0" />
  <field column="6" name="TRDPRC_1" />
  <field column="7" name="ACVOL_1" />
  <constant name="PROD_PERM" value="1" />
</stream>
```

This example identifies the first column of stream1 as its unique identifier or "key" column.

rfa

The **rfa** element provides information for configuring the Reuters side of the adapter, including an explicit reference to the Reuters-side configuration file.

Summary

```

adapter                                (required, limit one)
|----rfa                                (required, limit one)
|----subscriptions                      (required, limit one)
|   '----subscription                   (required)
|       '----stream                     (required)
|           |----name                   (required, limit one)
|           |----stale                   (optional)
|           |----field                   (required)
|           '----constant                (optional)

```

Parent

adapter

Children

None

Attributes

Name	Description	Requirement
serviceName	A service name that is included in the header of every update sent out by the Reuters OMM adapter	Optional
config	The absolute path and file name of the Reuters-side configuration file for publication (the sample file supplied with the adapter is at <code>\$ESP_RMDSOMM_HOME/config/rmdsomm.cfg</code>)	Required
sessionName	A reference to a session named defined in the Reuters-side configuration file for publication	Required
configDatabaseName	A reference to the Reuters database name	Optional
blankDate	A marker to use for blank <code>Date</code> fields	Optional
blankDouble	A marker to use for blank <code>Double</code> fields	Optional
blankInt32	A marker to use for blank <code>Int32</code> fields	Optional
blankInt64	A marker to use for blank <code>Int64</code> fields	Optional
blankMoney	A marker to use for blank <code>Money</code> fields	Optional

Name	Description	Requirement
blankString	A marker to use for blank String fields	Optional
blankTimestamp	A marker to use for blank Timestamp fields	Optional
enumFile	The full path name of the Reuters-supplied file that lists each enumerated type along with the range of values it can take	Optional (the default is the enum-Type definition)
fidFile	The full path name of the Reuters-supplied file that lists all of the valid FIDs	optional (the default is the RDMField Dictionary)

Notes

None

Example

```
<rfa serviceName="IDN_RDF"
    config="$ESP_RDMSOMM_HOME/config/rmdsomm.cfg"
    sessionName="Session1" configDatabaseName="RFA" />
```

This example points the Reuters OMM adapter to the Reuters-side configuration in the file `rmdsomm.cfg` key. The first five uncommented lines in this configuration file are:

```
\Connections\Connection_RSSL\connectionType = "RSSL"
\Connections\Connection_RSSL\hostName = "tigris.sybase.com"
\Connections\Connection_RSSL\rsslPort = "14002"
\Connections\Connection_RSSL\connectRetryInterval = 7000
\Sessions\Session1\connectionList = "Connection_RSSL"
```

The last of these lines implicitly defines a session name that is defined as the **sessionName** in the map file. The other three lines from `rmdsomm.cfg` key on this session name. This is how the value for **sessionName** ties this publication section of the map file to a configuration set in the `.cfg` file.

When the adapter publishes using this configuration, each update is identified with the **serviceName** "IDN_RDF."

stale

In a **stream** definition in an output adapter map file, the **stale** element identifies a column in the source stream for which the value changes from 0 to 1 if the stream goes stale.

Summary

```
adapter (required, limit one)
|----rfa (required, limit one)
'----subscriptions (required, limit one)
'----subscription (required)
```

```

'----stream          (required)
  |----name          (required, limit one)
  |----stale         (optional)
  |----field         (required)
  |----constant      (optional)

```

A stream is considered to have gone stale if, for example, one of the stream's data sources is no longer being updated.

Parent

stream

Children

None

Attributes

Name	Description	Requirement
column	A number that represents the position of the column with the secondary key value (the first column in the stream has the number 0)	Required
name	A string that identifies the stale column so that it may be mapped to a FID (published)	Optional

Notes

None

Example

```

<stream name="stream1" >
  <name column="0" />
  <stale column="3" name="ACVOL_1" />
  <field column="1" name="DSPLY_NAME" />
  <field column="4" name="BID" precision="47" />
  <field column="5" name="ASK" precision="0" />
  <field column="6" name="TRDPRC_1" />
  <field column="7" name="ACVOL_1" />
  <constant name="PROD_PERM" value="1" />
</stream>

```

This example identifies the third column of stream1 as its stale column. If the stale column is specified, the column value is published and the RIC is marked stale.

stream

In a subscription section in an output adapter map file, identifies the stream from which the adapter gets the data it publishes to RMDS.

Summary

```

adapter                                (required, limit one)
|----rfa                                (required, limit one)
|----subscriptions                      (required, limit one)
|   '----subscription                  (required)
|       '----stream                    (required)
|           |----name                  (required, limit one)
|           |----stale                  (optional)
|           |----field                  (required)
|           '----constant              (optional)
    
```

Parent

subscription

Children

Name	Requirement
name	One
stale	Optional
field	One or more
constant	Optional

Attributes

Name	Description	Requirement
exitOnStreamExit	This is a boolean attribute. When true, esp_rmdbomm terminates if the stream exits, Event Stream Processor exits, or the connection is lost.	Optional (default is false)
finalizer	This string specifies an action to take if the specified number of heartbeat milliseconds elapse without an event being published to Event Stream Processor.	Optional
heartbeat	This integer specifies how many milliseconds to wait without an event being published to Event Stream Processor before executing the finalizer action .	Optional

Name	Description	Requirement
name	The name of the stream from which the adapter receives the data it publishes on RMDS	Required
platformExitOnStream-Drop	This is a boolean attribute. When true, Event Stream Processor exits if this subscription drops its connection.	Optional (default is false)
platformQueueSize	The size, in bytes, of Event Stream Processor queue	Optional (default is 8000)

Notes

The value of the **name** attribute must be defined in Event Stream Processor project.

Any stream in Event Stream Processor project can map to only one **stream** section in the map file.

Example

```
<stream name="stream1">
  <name column="0"/>
  <field column="4" name="TRDPRC_1"/>
  <field column="9" name="BID" precision="5"/>
</stream>
```

This example configures Event Stream Processor to publish data from a stream named stream1.

subscription

The **subscription** element contains one or more instances of the **stream** element; enabling you to configure the adapter to receive data from one or more streams.

Summary

```
adapter                (required, limit one)
  |----rfa              (required, limit one)
  '----subscriptions   (required, limit one)
    '----subscription  (required)
      '----stream      (required)
        |----name      (required, limit one)
        |----stale     (optional)
        |----field     (required)
        '----constant  (optional)
```

The output adapter map file can contain two or more **subscription** sections. At runtime, the publishing mechanism for each **subscription** section is instantiated on a separate thread, which provides scalability.

Parent
subscriptions

Children

Name	Requirement
stream	One or more

Attributes

Name	Description	Requirement
name	A name for this subscription that appears in updates published on RMDS and in log file entries	Required

Notes
None

Example

```
<subscriptions>
  <subscription name="subscription1" >
    <stream name="stream1" >
      <name column="0"/>
      <field column="4" name="BID"/>
      <field column="5" name="ASK"/>
      <field column="6" name="TRDPRC_1"/>
      <field column="7" name="ACVOL_1"/>
      <constant name="PROD_PERM" value="1"/>
    </stream>
  </subscription>
</subscriptions>
```

This example configures the adapter to publish some columns from stream1 on Event Stream Processor using the name subscription1.

subscriptions

The **subscriptions** element contains one or more **subscription** elements.

Summary

```
adapter (required, limit one)
|----rfa (required, limit one)
'----subscriptions (required, limit one)
    '----subscription (required)
        '----stream (required)
            |----name (required, limit one)
            |----stale (optional)
            |----field (required)
            '----constant (optional)
```

Parent
adapter

Children

Name	Requirement
subscription	One or more

Attributes
None

Notes

Each **subscription** instance in this section defines one set of data that the adapter publishes to RMDS.

Example

See the example for an individual **subscription** instance.

Logging Facilities

The Reuters OMM adapter supports two different logging mechanisms.

In addition to its own logging mechanism, the Reuters OMM adapter can utilize Reuters-side logging. You can use both of these mechanisms to check the adapter's performance and diagnose problems.

You can configure these logs to be written to stderr, syslog, or both.

Adapter Logging

The Reuters OMM adapter supports the same options for logging as the Event Stream Processor.

The **-d** option sets the debug level (0=emergency messages only, 7=all messages).

The **-l** option tells the adapter to write log messages to stderr, syslog, both, or neither. If you use the **-l** option to direct adapter log messages to stderr, you may also want to redirect stderr to a file.

The name attribute of the **publication** element in the input adapter map file specifies a descriptive text string that is logged to help identify how the adapter was configured. For example, lines 3–6 of `subexample.xml` specify the **publication** element for a subscribing instance of the Reuters OMM adapter, as follows:

```
<publication
  name="RMDS OMM Adapter"
  retryInterval="5"
/>
```

CHAPTER 2: Adapters Supported by Event Stream Processor

As the adapter connects with and interacts with Event Stream Processor, this configuration causes the adapter to write log messages similar to:

```
(0.123) @1 INFO: Configuring publication with name RMDS Adapter exp
```

The first two fields are the timestamp (in seconds since start-up) and the thread number, respectively. The base time for the timestamp, along with other information, is written to the log file on startup as shown in the example below. To convert the timestamp to a date and time, simply add the number of seconds to the base time.

```
(63359098041.768) @1 NOTICE:Base time is 10/08/08-17:27:21
(0.001) @1 NOTICE:insta-a sub -c cimtest:-- -d 7
-f /home/sybase/support/1.0.3/ReutersOMMAdapter/marketprice.map.xml
-l 1 -p tigris:12192 -P 1
(0.001) @1 NOTICE:pid=28649
(0.001) @1 DEBUG:Using ESP_RMDSOMM_SUBSCRIBE_DEBUG_LEVEL=711/
i86pc_64_spro/bin/rmdsomm version:
1.0.3a-alpha_r18674M
```

Page Data and Partial Page Updates

Some Reuters data comes as pages which use the partial page format. Each page consists of multiple lines; initially sent as a snapshot. Page data is supported without any special configuration. The following extract from an adapter log file shows the delivery of the initial page image (which is highlighted).

```
(27.729) @6 INFO:Publishing VOD.mGBPd 21 of 21 on stream1 as UPSERT
_ITEM_NAME_ STRING: VOD.mGBPd
_SERVICE_NAME_ STRING: IDN_RDF
_SEQUENCE_NUMBER_ INT32: 1
_ITEM_STALE_ INT32: 0
ROW80_1 STRING: VOD.mGBPd SI Quote Publication
ROW80_2 STRING:
ROW80_3 STRING: DATE:03/07/2008 Time:11:09
ROW80_4 STRING:
ROW80_5 STRING: Time Venue SI Bid Size Bid Price Ask Price Ask Size
Status
ROW80_6 STRING: ==== ===== == ===== ===== =====
=====
ROW80_7 STRING: 110937 GSILGB2XXXX GSIL 1 150.9000 150.9500 1 OPEN
ROW80_8 STRING: 070021 SBILGB2LXXX CITI OPEN
ROW80_9 STRING: 110909 CSFBGB2LXXX CSFB 329 150.7000 151.1500 329
OPEN
ROW80_10 STRING: 110942 DEUTGB22ZEQ DBBL 528 150.6500 151.2000 527
OPEN
ROW80_11 STRING: 110946 ABNAGB22XXX ABNV 483306 150.9000 150.9500
483306 OPEN
ROW80_12 STRING: 110936 UBSWGB2LEQU UBSI 1 149.7682 152.1325 1 OPEN
ROW80_13 STRING: 110828 SBUKGB21XXX CITI 20600 150.9000 151.0000
20600 OPEN
ROW80_14 STRING: 110937 SLIIGB2LXXX LEHM 3750 150.9000 150.9500 15
OPEN
ROW80_15 STRING:
ROW80_16 STRING:
ROW80_17 STRING:
(27.730) @6 DEBUG:Immediate flush for low latency; opcode=p
```


Each line of the page has its own FID to facilitate line-oriented deltas to the page. The adapter parses the partial page updates from Reuters and produces strings like the ones shown highlighted in the following extract from an adapter log file.

```
(49.934) @6 DEBUG:Processing update for VOD.mGBPd from service
IDN_RDF
(49.934) @6 INFO:Publishing VOD.mGBPd 4 of 21 on stream1 as UPSERT
_ITEM_NAME_ STRING: VOD.mGBPd
_SEQUENCE_NUMBER_ INT32: 2
ROW80_3 STRING: off:78 size:2 value:10
ROW80_11 STRING: off:2 size:3 value:101
(49.934) @6 DEBUG:Immediate flush for low latency; opcode=p
(50.315) @6 DEBUG:Processing update for VOD.mGBPd from service
IDN_RDF
(50.315) @6 INFO:Publishing VOD.mGBPd 3 of 21 on stream1 as UPSERT
_ITEM_NAME_ STRING: VOD.mGBPd
_SEQUENCE_NUMBER_ INT32: 3
ROW80_11 STRING: off:5 size:1 value:7
(50.315) @6 DEBUG:Immediate flush for low latency; opcode=p
```

The first update in the example is to write the 2-character string 10 at an offset of 78 characters in the line of the page which contains the data from the ROW80_3 FID. The second update in the example is to write the 3-character string 101 at an offset of 2 characters in the line of the page which contains the data from the ROW80_11 FID. The third update in the example is to write the 1-character string 7 at an offset of 5 characters in the line of the page which contains the data from the ROW80_11 FID. Thus, updates for page data are very concise.

Modifying Log Entry Format

You can modify the default format of log entries in two ways.

Set the environment variable `ESP_RMDS_SUBSCRIBE_SYMBOL_FORMAT` to 1 to configure your system to log messages that show what values flow to the Event Stream Processor on a single line rather than the default multiline format. When messages are written to a log file, this can make it easier to scan for specific items.

Use the `-P` option to the `esp_rmddsomm` command to specify the number of decimal places that appear on output for double type variables.

By default, log messages that show what values flow to Event Stream Processor are written in multiline format as shown:

```
(38079.526) @2 INFO:Publishing VOD.mGBPd 3 of 9 on stream1 as UPSERT
_ITEM_NAME_ STRING: VOD.mGBPd
_SEQUENCE_NUMBER_ INT32: 953
ROW80_7 STRING: off:53 size:2 value:45
```

If you set the environment variable `ESP_RMDS_SUBSCRIBE_SYMBOL_FORMAT` to 1 these messages are written in single-line format as shown:

```
(17.794) @5 DEBUG:stream1 p values: _ITEM_NAME_=VOD.mGBPd
_SEQUENCE_NUMBER_=2
ROW 80_3=off:78 size:2 value:20
```

CHAPTER 2: Adapters Supported by Event Stream Processor

The `-P` option can alter the manner in which double datatype variables appear, as shown by `ask` and `last` are in the following example. This affects only the way variables appear; it does not alter the contents.

```
<RowDefinition id="omm_RowDef">
<Column name="symbol" datatype="string" />
<Column name="service" datatype="string" />
<Column name="seq" datatype="integer" />
<Column name="stale" datatype="integer" />
<Column name="bid" datatype="money" />
<Column name="ask" datatype="double" />
<Column name="last" datatype="double" />
<Column name="volume" datatype="integer" />
<Column name="when" datatype="timestamp" />
</RowDefinition>
```

If you accept the default precision, variables of type double (for example, `ASK` in the following example) are written with three digits to the right of the decimal

```
(5.089) @5 INFO:Publishing EURJPY= 7 of 9 on stream1 as UPSERT
(5.090) @5 DEBUG:stream1 p values: _ITEM_NAME_=EURJPY=
_SEQUENCE_NUMBER_=1 _ITEM_STALE_=0 BID=137.4800 ASK=137.530
ACVOL_1=0
ACTIV_DATE+TIMACT=2008-10-06T21:07:00.000 (1223327220000)
```

If you specify the option `-P 7` when enter the `esp_rmdsomm` command, variables of type double (for example, `ASK` in the following example) are written with seven digits to the right of the decimal. Variables of other types are not affected.

```
(4.913) @5 INFO:Publishing EURJPY= 7 of 9 on stream1 as UPSERT
(4.913) @5 DEBUG:stream1 p values: _ITEM_NAME_=EURJPY=
_SEQUENCE_NUMBER_=1 _ITEM_STALE_=0 BID=137.5200 ASK=137.5700000
ACVOL_1=0 ACTIV_DATE+TIMACT=2008-10-06T20:55:00.000 (1223326500000)
```

Reuters Logging

Turn Reuters logging on or off using the Reuters-side configuration file.

You can configure the Reuters OMM adapter's interface to RMDS to write to a logging facility. In the Reuters-side configuration file (`rmdsomm.cfg` is the one provided with the adapter), you can turn logging on or off and specify a path and file name of the log file. The Reuters interface also supports a set of "message files."

The Reuters-side configuration file contains a set of configuration entries for the Reuters "Logger" facility.

```
\Logger\AppLogger\fileLoggerEnabled           = true
\Logger\AppLogger\fileLoggerFilename         = "rfasub.{p}.log"
```

This configuration turns on Reuters logging for the Reuters OMM adapter. The log messages are written to the `rfasub.PID.log` file, where **PID** is the adapter's process ID.

The first line in this set, `\Logger\AppLogger\windowsLoggerEnabled = false`, pertains to a Windows logging facility that is not supported for the Reuters OMM adapter.

These example lines are from `rmdsomm.cfg`, the file that configures an adapter that subscribes to RMDS.

The same file contains configuration entries for Component Loggers, as follows:

```
\Logger\ComponentLoggers\Connections\messageFile =\  
  "config/messages/RFA7_Connections.mc"  
\Logger\ComponentLoggers\Adapter\messageFile =\  
  "config/messages/RFA7_Adapter.mc"  
\Logger\ComponentLoggers\SessionCore\messageFile =\  
  "config/messages/RFA7_SessionLayer.mc"  
\Logger\ComponentLoggers\SSLED_Adapter\messageFile =\  
  "config/messages/RFA7_SSLED_Adapter.mc"
```

Log Messages

Examples of typical entries from the adapter log file.

The actual format and working of the log messages, as well as the nature of the events logged and the log levels associated with these events, may change in subsequent releases of the adapter.

- **Message:** – NOTICE:Item BARC.VX is closed: No Quality of Service is available to process subscription, timeout expired
- **Cause:** – the value for the Reuters user name in the Reuters config file is incorrect (verify the case-sensitivity) or the Reuters Service name in the map file is incorrect.
- **Message:** – DEBUG: Immediate flush for low latency
- **Cause:** – data received from RMDS is being sent to Event Stream Processor immediately.
- **Message:** – NOTICE:XMLRPC ERROR-116: The connection to the server could not be established. Please make sure the server is up, and check the specified host name/port, user name/password, and encryption settings. If a host name is specified, make sure that it can be resolved through a DNS lookup. (5.092) @1 INFO:Could not connect to SP; (tigris: 12190 cimtest) will retry in 5 seconds.
- **Cause:** – cannot connect to the server running Event Stream Processor.
- **Message:** – Ignoring market data event because no significant fields updated
- **Cause:** – the adapter received data from Reuters, but none of the fields were of interest to Event Stream Processor stream, so no data was sent.

CHAPTER 2: Adapters Supported by Event Stream Processor

- **Message:** – ERROR: Error publishing: PUBLICATION ERROR-442: The send method of this publication object failed.
- **Cause:** – connection to Event Stream Processor unsuccessful during a message transmission.
- **Message:** – ERROR:Mismatch between platform stream (9 columns) and adapter (31 columns for stream: stream1)
- **Cause:** – the number of columns defined in the adapter did not match the number of columns in the stream.
- **Message:** – WARNING: Event Stream Processor down, dropping all subscriptions
followed by multiple iterations of a message similar to:
DEBUG: Unsubscribing item: EUR= service: IDN_RDF
- **Cause:** – lost connection to Event Stream Processor. Stopping subscriptions to RMDS data since the adapter has nowhere to put it.
- **Message:** – WARNING: Discarding data rec'd after unsubscribe
- **Cause:** – before the adapter shut off the subscription, additional data arrived. The data has been discarded because there is no connection to Event Stream Processor.
- **Message:** – DEBUG: Processing update for EUR= from service IDN_RDF
- **Cause:** – an update for RIC "EUR=" on service named "IDN_RDF" has arrived.
- **Message:** – WARNING: Event Stream Processor down, dropping all subscriptions
followed by numerous repetitions of:
DEBUG: Unsubscribing item: EUR= service: IDN_RDF
- **Cause:** – lost connection to Event Stream Processor. Stopping subscriptions to RMDS data since the adapter has nowhere to put it.
- **Message:** – WARNING: Discarding data rec'd after unsubscribe
- **Cause:** – before the adapter shut off the subscription, additional data arrived. The data has been discarded, because there is no connection to Event Stream Processor.
- **Message:** – EMERGENCY: Fatal Error at line 0, column 0 of config file: An exception occurred! Type:RuntimeException, Message:The primary document entity could not be opened. Id=/home/sybase/adapter/trunk/src/ReutersAdapter/xxsubexample.xml
- **Cause:** – specified configuration file is unavailable.
- **Message:** – EMERGENCY: Fatal Error at line 0, column 0 of config file: An exception occurred! Type:RuntimeException,

Message:The primary document entity could not be opened.
 Id=/home/sybase/adapter/trunk/src/ReutersAdapter/
 xxsubexample.xml

- **Cause:** – specified config file is unavailable.

RTView Adapter

The Sybase Event Stream Processor RTView adapter is an external adapter that streams data from Event Stream Processor to the RTView® Enterprise Dashboard. RTView Enterprise software from SL Corp. is required to operate this adapter.

While this document provides information on configuring the RTView software for use with the adapter, you should also consult your SL Corp. documentation for complete details and the most up-to-date information.

Datatype Mapping for the RTView Adapter

Event Stream Processor datatypes map to RTView datatypes.

Event Stream Processor Datatype	SL RTView Datatype
boolean	boolean
integer	integer
long	long
float	double
date	date
timestamp	date
string	string
money, money(1) .. money(15)	double
binary	string
interval	long
bigdatetime	date

Installing the RTView Adapter

To install the RTView adapter, unpack the RTView adapter files and set the environment variables for the adapter and the RTView software.

Prerequisites

- Install either version 5.8 or 5.9 of the Enterprise RTView software from SL Corporation on the client machine.
- Install Java Software Development Kit 1.6 (or higher) on the client machine.
- Set the `JAVA_HOME` environment variable to the root directory of the installation.
- Set the `ESP_HOME` environment variable.

Task

1. Create an environment variable called `RTVIEWADAPTER_HOME` and set its value to the folder `$ESP_HOME \adapters\rtview`.
2. Verify that the `RTV_HOME` environment variable is set to the location of the enterprise RTView installation. It should be set automatically during installation.
3. Add the RTView `lib` and `bin` folders to the `PATH` environment variable. For example, update `PATH` to `$RTV_HOME/bin;$RTV_HOME/lib;$PATH`.

Configuration: Creating and Updating a Sybase Connection

Create and update the `ESOPTIONS.ini` configuration file with connection information for the Server.

You can create multiple connections to the Server. Each connection has a specific server, host, and properties. You require at least one connection to the Server.

1. In the Display Builder, select **Tools > Options**.
2. In the left pane of the Application Options window, choose **ESP**.
3. In the ESP Connections tab, click **Add**.
4. To modify the properties of an existing connection, double-click the connection.
5. Fill in the appropriate connection information and click **OK**.
6. Click **Apply**, then **Save** to save the connection information in the `ESOPTIONS.ini` configuration file. When asked if you want to save the configuration file to the `lib` folder in the adapter installation directory, click **No**. This ensures your connection information is applied only to the current project.

Next

After you have created and edited the connection, restart either the RTView Display Builder or the Server for the changes to take effect.

Event Stream Processor Parameters

Parameters that you can specify within the `ESOPTIONS.ini` configuration file to create a connection to Event Stream Processor.

Parameter	Type	Description
authType	choice	(Required) Specifies the method used to authenticate to Event Stream Processor. No default value. <ul style="list-style-type: none"> • UserPassword – user and password parameters are required. • ServerRSA – user, keyStore, and keyStoreFile parameters are required. • None.
projectUri	string	(Optional) Specifies the total project uri to connect to Event Stream Processor cluster. For example, <code>esp://hostname:port/workspace/project</code> . No default value.
keyStore	string	(Optional) Specifies the location of the RSA keystore, and decrypts the password value. Required if authType is set to ServerRSA. No default value.
keyStorePassword	string	(Optional) Specifies the keystore password, and decrypts the password value. Required if authType is set to ServerRSA. No default value.
user	string	(Optional) Specifies user name required to log in to Event Stream Processor. Required for any authentication scheme other than none (see authType). No default value.
password	string	(Optional) Specifies the password required to log in to Event Stream Processor. Required for User-Password authentication scheme (see authType). No default value.

Parameter	Type	Description
isEncrypted	string	(Optional) Specifies whether password is encrypted. Valid values are true or false. If set to true, password is an encrypted field. This ensures that the Server recognizes the password as encrypted text and is able to decrypt the password at runtime. Default value is true.
getBase	string	(Optional) Specifies whether the Server sends existing data in stream at the time the subscription is set up. Valid values are true or false. If set to true, the Server does not send existing data in the stream. Default value is false.
droppable	string	(Optional) Specifies whether the Server drops this connection if that client cannot keep up. If set to true, the Server drops the connection. Default value is false.
lossy	string	(Optional) Specifies whether the Server may discard records if the client cannot keep up. If set to true, the Server discards records. Default value is false.
shineThrough	string	(Optional) Specifies whether Event Stream Processor sends data for fields which have not changed. If set to true, the Server does not send data. Default value is false.
refreshInterval	integer	(Optional) Specifies the pulse interval. Event Stream Processor consolidates data and sends it periodically at intervals as specified in milliseconds. A value of 0 or less disables pulsing. No default value.
dateFormat	string	(Optional) Specifies date format. Default value is YYYY-MM-DDHH24:MI:SS:FF.
timestampFormat	string	(Optional) Specifies timestamp format. Default value is YYYY-MM-DDHH24:MI:SS:FF.
delimiter	string	(Required) Used in the publisher command line. Default value is ##.
defaultconnection	string	(Required) Specifies default connection settings used to connect to the Server. Default value is conn1.

Parameter	Type	Description
retryInterval	integer	(Required) Specifies how long to wait to reconnect to the Server if the connection is broken. Default value is 0.
pollinterval (seconds)	integer	(Required) Specifies how long to wait to poll data. Default value is 0.

Operation

Once you have installed the RTView adapter, you can begin using the Display Builder and the Display Viewer.

Starting the RTView Display Builder

To build and run dashboard projects, start the Display Builder from the command line.

Prerequisites

Start the Server.

Task

Running the Display Builder from the adapter installation folder in the command line ensures that the Builder links to the Server upon start-up.

Sybase recommends that you place each dashboard project in its own folder. You can then start the Display Builder from this folder.

1. To start a new dashboard project, create a new folder for it. To open an existing project, select **Start > Run**.
2. Start the project in the RTView Display Builder by typing:

```
%RTVIEWADAPTER_HOME%\bin\start_builder.bat <project_filepath>
[<rtv_file_name>.rtv]
```

- <project_filepath> is the absolute file path of the project folder.
- <rtv_file_name> is the name of an existing dashboard. When creating a new .rtv file, do not supply a file name: the Display Builder opens into a blank dashboard called unnamed.rtv, which you can then save with a desired name into the new dashboard project folder created in step 1.

Note: On Windows, if you type the start builder command and use only the project folder argument without a file name, it looks for a file named "<projectfolder>.rtv". If such a file does not exist in the folder, you see a message that the file cannot be opened. Click **OK** to launch the builder. This is a known RTView issue.

Starting the RTView Display Viewer

To begin viewing runtime data, start the Display Viewer from the command line.

Prerequisites

Start the Server.

Task

Running the Display Viewer from the adapter installation folder in the command line ensures that the Viewer links to the Server upon start-up.

Sybase recommends that you place each dashboard project in its own folder. You can then start the Display Viewer from this folder.

1. To start a new dashboard project, create a new folder for it. To open an existing project, select **Start > Run**.
2. Start the project in the RTView Display Viewer by typing:

```
%RTVIEWADAPTER_HOME%\bin\start_viewer.bat <project_filepath>  
<rtv_file_name>.rtv
```

- <project_filepath> is the absolute file path of the project folder.
- <rtv_file_name> is the name of the .rtv file of the dashboard. You need to specify this to start the Viewer.

Creating Shortcuts for Dashboard Projects

Starts the Display Builder or Viewer from a convenient location.

The shortcut starts the Builder or Viewer and simultaneously opens a specified dashboard project.

1. At the location where you want to create the shortcut, select **File > New > Shortcut** from the menu bar, or right-click and select **New > Shortcut**.
2. In the Create Shortcut wizard, assign the shortcut a name, then enter
%RTVIEWADAPTER_HOME%\bin\start_builder.bat
<project_filepath> [<rtv_file_name>.rtv] as the location of the item.
<project_filepath> is the absolute file path of the project folder.
<rtv_file_name> is the name of the .rtv file of the dashboard you want to open.
Click **Next**.
3. Right-click the shortcut and select **Properties**.
4. In the Properties window, set the Run field to **Minimized**.
5. Repeat steps 2—4 to create a corresponding shortcut that starts the dashboard project in the Display Viewer.

Enter %RTVIEWADAPTER_HOME%\bin\start_viewer.bat
 <project_filepath> <rtv_file_name>.rtv as the location of the item.

Dashboard Objects and Data Streams

You can connect most RTView dashboard objects to Event Stream Processor data streams. With this connection, dashboard objects can receive real-time data from streams.

There are two approaches for connecting to streams, depending on the type of stream:

- If the stream produces updates and deletes against keyed entries, first set up an intermediate object known as a cache. You can then connect the dashboard objects.
- If the stream contains insert elements, such as a timeseries, connect the dashboard object directly to the stream.

Creating a Cache

Use the RTView Builder to create a cache in a separate .rtv file, then import the file into the main dashboard file.

A cache is a datasource that allows users high-speed analytic processing of real-time data and the comparison of current real-time values against historical data. It is an intermediate datasource when connecting a dashboard object to an Event Stream Processor data stream that produces updates and deletes against user-entered values. From the RTView Builder:

1. Select **File > New** to create a new .rtv file.
2. Select **Tools > Caches**, then click **Add** in the Caches tab at the bottom of the main Display Builder window.
3. Enter a name for the new cache and set its type to **Table**.
4. Edit the cache properties:

Property	Procedure
valueTable	<ol style="list-style-type: none"> 1. Right-click valueTable and select Attach To Data > ESP. 2. Select the name of the stream that the cache subscribes to. 3. (Optional) Choose specific stream columns; however, if you select specific columns, you must select the primary-key columns.
indexColumnNames	<ol style="list-style-type: none"> 1. Click the ellipsis (...) beside the indexColumnNames field. 2. Specify the key columns for the stream, separating the column names with a semicolon. <p>Note: The column names are case-sensitive.</p>

Property	Procedure
rowsToDeleteTable	<p>rowsToDeleteTable is a data attachment that removes selected rows from the cache tables. Rows are removed if their index column values match those of a row in the table data coming from this attachment.</p> <ol style="list-style-type: none"> 1. Right-click rowsToDeleteTable and select Attach To Data > ESP. 2. Event Stream Processor does not have a rowsToDeleteTable, so you must use a virtual table name. For example, to use the Positions stream name, use !Positions for the rowsToDeleteTable property.
maxNumberOfRows	<ul style="list-style-type: none"> • Specify the number of rows of historical data to save. The default value is zero. Anything larger than zero enables storage of historical data. For every key value, N rows of history are maintained, where N is the number of rows specified.

5. Save the file.
6. Import this file to the main dashboard file. From the main dashboard file:
 - a) Select **Tools > Options > Caches**.
 - b) Click **Add** and select the cache created in step 3.
 - c) Click **Apply**, then **OK**.

Note: If you make any changes to the cache after importing it to the main display file, select **Options > Caches > Refresh Selection**.

7. Save, then close, the file.
8. Repeat steps 1 to 7 for every cache you are creating.

Example: Attaching an Object to a Cache

Attach a dashboard table object to a previously created cache.

You cannot directly connect dashboard objects to streams that make updates or deletes against keyed entries. Connect a cache to the stream, then connect the object to the cache.

Attach any object to streams, either directly or through caches, by setting the object's value property under the Data heading in the Object Properties pane. For a table object, this property is called valueTable.

In this example, first attach a table to a dashboard, then use the valueTable property to attach the table to a previously created cache that connects to an Event Stream Processor stream.

1. From the **Tables** tab in the Object Palette, select a table, then click the canvas to add the table object to the dashboard.
2. Import the cache to the dashboard project:
 - a) Select **Tools > Options > Caches**.
 - b) Click **Add** and select the cache.
 - c) Click **Apply**, then **OK**.

3. Right-click the valueTable property of the table object, and go to **Attach To Data > Cache**.

This opens a dialog box to configure the datasource.

4. Choose the cache from step 2.
5. Choose **Current** in the Table field and select the columns you want to view.
6. (Optional) Select Filter Rows to Basic or Advanced to view a subset of the data.
7. Click **Apply**, then **OK**.

Example: Attaching an Object to a Stream

Attach a Dashboard table object directly to a stream that contains only inserts.

Attach any object to streams, either directly or through caches, by setting the object's value property under the Data heading in the Object Properties panel. For a table object, this property is called valueTable.

1. From the **Tables** tab in the Object Palette, select a table, then click the canvas to add the table object to the dashboard.
2. Right-click the valueTable property of the table object and select **Attach To Data > ESP**.
3. Select the connection to use, and the table and columns you want to view.
4. If desired, filter the rows and columns of the stream to view a subset of the data.

Example: Creating a Function

Create a function that returns a list of table values to populate a listbox object (labelled obj_c1tlb).

Functions allow you to automate common calculations, such as taking the average value of a table column and adding the values of multiple data attachments.

There are several functions grouped into two categories, Scalar Functions and Tabular Functions, that act on scalar and tabular data, respectively.

From the Display Builder:

1. Select **Tools > Functions**, or click the **Functions** tab at the bottom of the Display Builder window, and click **Add**.
2. Enter a name for the function and choose an appropriate function type. For example, choose Count Unique Values to return a list of unique values from a column in a table or stream. If you select this function, you are prompted to specify a table or stream column.
3. Right-click the Table Column field and select **Attach To Data > ESP**. Choose a connection to the Event Stream Processor stream name and a column defined in the stream.
4. Create a listbox object in the Controls tab of the Object Palette.
5. Right-click the **listValues** property of the listbox object properties, and select **Attach To Data > Functions**. Choose the function you created in step 2 and select **Value** in the Columns field to bind the function to the listbox object.

Whenever a new value appears in a column in the stream that the function has been attached to, that value also appears automatically in the listbox object.

6. Set the `selectedValue` and `varToSet` properties of the listbox object so the selections made in the listbox can be used elsewhere, such as for publishing to Event Stream Processor.

Publishing to Event Stream Processor

Add user control objects to a dashboard project so you can actively interact with Event Stream Processor streams.

The RTView adapter supports publishing data to the Event Stream Processor from a dashboard. The dashboard sends an event through the RTView adapter to an input stream on the Server.

1. Add one or more control objects to the dashboard.
 - Add a control for each field you want to set. You can use input boxes, picklists, listboxes, checkboxes, and buttons.
 - a) Use the Tools menu in the Display Builder to create a variable for each field.
 - b) Attach each control to a variable.
 - c) Add variables by selecting **Tools > Variables** and specifying the name, initial value, and type of the variable.
 - d) (Optional) Select **Tools > Functions** and create a function to determine the user choices in the control and attach the control to that function.
 - e) (Optional) Link the property of other objects on the dashboard to the control.
2. Associate an action command with the Control Object by right-clicking **actionCommand** under the Interaction category, and selecting **Define Command > ESP**. Enter a publish command:

```
conn_name.publish ## opcode ## stream_name ## col_value_1 ##
col_value_2 ...
```

Parameter	Description
##	The argument delimiter, which you can set to anything. Default value is ##. Select Tools > Options , select ESP on the left pane, then click Add button. The delimiter must have a space before and after it.
conn_name	A predefined connection to be used for publishing.
opcode	The operation to perform. Valid values are: <ul style="list-style-type: none"> • 1 — INSERT • 3 — UPDATE • 5 — DELETE • 7 — UPSERT • 13 — SAFE DELETE

Parameter	Description
stream_name	The name of the target stream.
col_value_...	<p>The value of a column. The number of column values must equal the number of columns in the target stream. You can specify NULL values by entering two single quotation marks with nothing between them (for example, ") . Default value is " .</p> <p>The RTView Viewer publishes empty fields as two single quotes ("). Because the default nullValue property is also ", a NULL is inserted into Event Stream Processor for corresponding columns.</p> <p>To specify an empty string, change the Null Value property in the Add ESP Connection window to a different value. For datatypes other than string, two single quotation marks without a space between them (for example, ") always represents a NULL value.</p>

- Specify date and timestamp values in the same format as in the Date and Timestamp format properties in the Add ESP Connection window. This is the same format specification the Java SimpleDateFormat object uses.
- If the publish command is successful, there is no response. If the command does not succeed, you see an appropriate error message.

Running the Publisher Example

Use the RTView Display Viewer to run the provided publisher example that comes with the RTView adapter.

1. Start Event Stream Processor with the provided `rtviewadapter.ccx` model, located in `%RTVIEWADAPTER_HOME%\examples`, and use port 19011.

Operating System	Step
Windows	<ol style="list-style-type: none"> 1. Enter <code>cd %RTVIEWADAPTER_HOME%\examples</code>. 2. Start the example cluster: <code>start_server_cluster.bat</code> 3. Add project to the cluster, and start it on the cluster: <code>start_project.bat</code>
UNIX	<ol style="list-style-type: none"> 1. Enter <code>cd \$RTVIEWADAPTER_HOME/examples</code>. 2. Start the example cluster: <code>start_server_cluster.sh</code> 3. Start the project on the cluster: <code>start_project.sh</code>

2. Start the RTView Display Viewer from the command line:

Operating System	Step
Windows	%RTVIEWADAPTER_HOME%\bin\start_viewer.bat %RTVIEWADAPTER_HOME%\examples publisher.rtv
UNIX	\$RTVIEWADAPTER_HOME/bin/start_viewer.sh \$RTVIEWADAPTER_HOME/examples publisher.rtv

Note: You can start the RTView Display Builder by replacing the **start_viewer** command with the **start_builder** command.

3. Follow the on-screen instructions.
4. Verify that data has been successfully published to the Server.

Operating System	Step
Windows	cd %RTVIEWADAPTER_HOME%\examples run_sub.bat
UNIX	cd \$RTVIEWADAPTER_HOME/examples run_sub.sh

Running the Subscriber Example

Use the RTView Display Viewer to run the provided subscriber example that comes with the RTView adapter.

1. Start the Event Stream Processor with the provided `rtviewadapter.ccx` model located in `%RTVIEWADAPTER_HOME%\examples`, and use port 19011.

Operating System	Step
Windows	<ol style="list-style-type: none"> 1. Enter <code>cd %RTVIEWADAPTER_HOME%\examples</code> 2. Start the example cluster: <code>start_server_cluster.bat</code> 3. Add project to the cluster, and start it on the cluster: <code>start_project.bat</code>
UNIX	<ol style="list-style-type: none"> 1. Enter <code>cd \$RTVIEWADAPTER_HOME/examples</code> 2. Start the example cluster: <code>start_server_cluster.sh</code> 3. Start the project on the cluster: <code>start_project.sh</code>

2. Start the RTView Display Viewer from the command line:

Operating System	Step
Windows	%RTVIEWADAPTER_HOME%\bin\start_viewer.bat %RTVIEWADAPTER_HOME%\examples subscriber.rtv
UNIX	\$RTVIEWADAPTER_HOME/bin/start_viewer.sh \$RTVIEWADAPTER_HOME/examples subscriber.rtv

Note: You can start the RTView Display Builder by replacing the **start_viewer** command with the **start_builder** command.

- Use the **esp_convert** and **esp_upload** utilities to load input .xml data, convert XML data to stream data, and feed this data into target streams.

Operating System	Step
Windows	cd %RTVIEWADAPTER_HOME%\examples run_loaddata.bat
UNIX	cd \$RTVIEWADAPTER_HOME/examples run_loaddata.sh

- (Optional) Verify that data is successfully loaded into streams.

Operating System	Step
Windows	cd %RTVIEWADAPTER_HOME%\examples run_sub.bat
UNIX	cd \$RTVIEWADAPTER_HOME/examples run_sub.sh

- Observe the stream data that displays in tables.

Known Limitations

Learn about the known limitations of the RTView adapter.

- To modify a connection that is already connected to the Server, restart either the Builder or the Server.
- You cannot publish values containing a period. As a workaround for double and money types, the adapter lets you type a comma for a decimal point instead of a period. There is no workaround for `string` datatypes.
- Because the RTView `double` datatype maps to the `money` datatype in Event Stream Processor, you may lose precision for data that has more than 15 digits.

CHAPTER 2: Adapters Supported by Event Stream Processor

- RTView adapter does not handle microsecond precision in their date datatype. As a result, the RTView date datatype maps to the bigdatetime datatype in Event Stream Processor, with millisecond precision. This is an RTView limitation to be resolved by SL Corporation.
- In the Add ESP Connection window, if you click **OK** before entering the connection name, you see an empty box.

TIBCO Rendezvous Adapter

Adapter type: tibcorvplugin. The Sybase Event Stream Processor TIBCO Rendezvous adapter subscribes to and publishes data from Event Stream Processor to the Rendezvous server.

The Rendezvous adapter:

- Connects to a Rendezvous server, opens sessions, subscribes and unsubscribes to subjects
- Translates Rendezvous messages into Event Stream Processor records and vice-versa
- Receives messages from and publishes messages to the Rendezvous server

See also

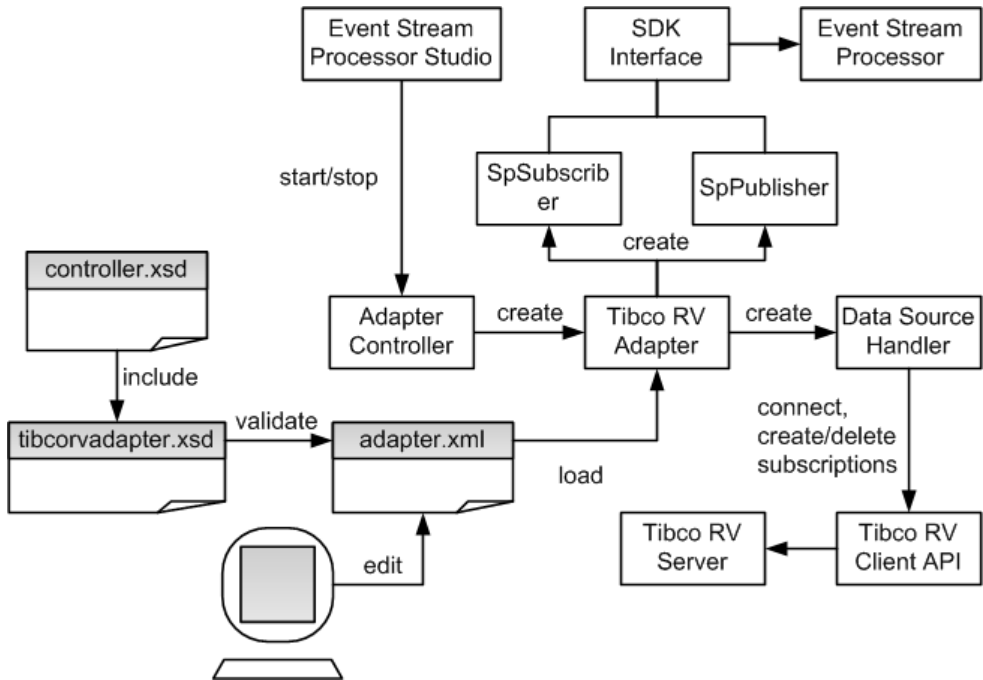
- *Chapter 5, Guaranteed Delivery* on page 543

Control Flow

The adapter loads its configuration from a file (for example, `adapter.xml`), and validates it against the adapter schema (`tibcorvadapter.xsd`), which includes the API-wide controller schema (`controller.xsd`).

You cannot edit schemas.

Figure 11: TIBCO Adapter Control Flow



The Adapter Controller creates an instance of the adapter which then receives and executes user commands. The Adapter Controller can execute **start**, **stop**, and **status** commands.

Start Command

The **start** command configures and starts the adapter command and control interface.

The Data Source Handler connects to and initiates a session with the Rendezvous server using the Rendezvous Client API. The SpSubscriber and SpPublisher components connect to Event Stream Processor using the Pub/Sub interface. SpSubscriber starts listening to the outbound streams and SpPublisher is ready to publish data to inbound streams.

If the **start** command is executed when there is a running instance of the adapter, it is ignored and a warning is sent that the adapter is already running.

See also

- *Starting the TIBCO Rendezvous Adapter* on page 501

Stop Command

The **stop** command disconnects the SpPublisher and SpSubscriber from Event Stream Processor, causes the Data Source Handler to close the session and disconnect from the

datasource, causes the Adapter Controller to stop listening to user commands, and terminates the adapter process.

If the **stop** command is executed when there is no instance of a running adapter, the command is ignored and a warning is sent.

See also

- *Stopping the TIBCO Rendezvous Adapter* on page 502

Status Command

The **status** command reports the adapter status, and the Adapter Controller prints out its status: either running or stopped.

See also

- *Checking the TIBCO Rendezvous Adapter Status* on page 502

Data Streams

The adapter stores each Rendezvous message in a stream record.

A single stream may store messages on different subjects. The subject is stored in a mandatory column Subject. The rest of the columns correspond to fields in Rendezvous messages.

Ensure the names of the stream columns are identical to the names of the corresponding scalar fields in Rendezvous messages. In case of message-type fields, the columns adhere to the following naming convention:

<message type field name>_<field name>

The adapter supports embedded messages of arbitrary depth. Columns unrelated to Rendezvous messages are not allowed, and fields of array type are not supported.

The Client and Date columns correspond to scalar fields. Trade is an embedded message which contains two fields: Price and Volume.

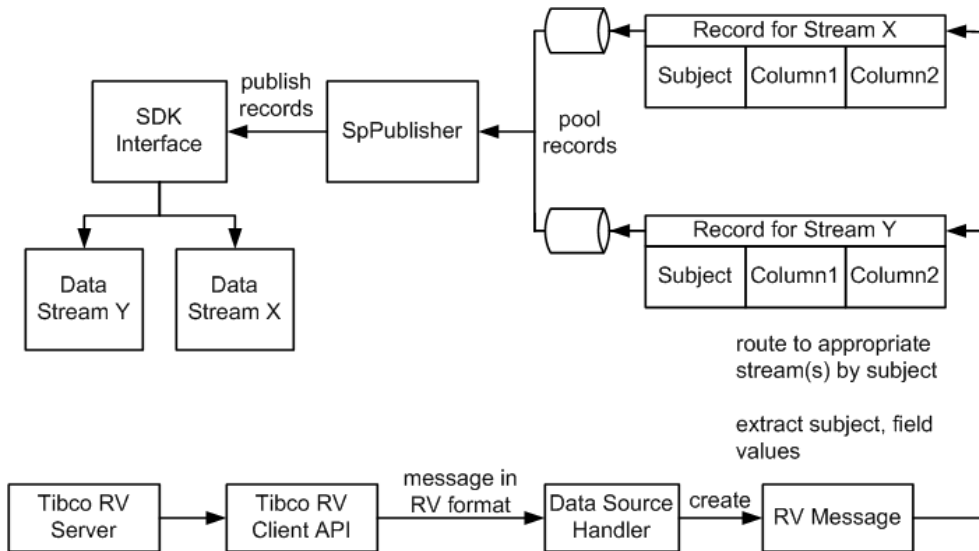
Table 12. Sample data stream

Sub-ject	Client	Date	Trade_Price	Trade_Volume
MySub-ject	UBS	2008-03-13T08:19:30	34.7	6000

Message Flow

The start command initiates the message flow through the adapter.

This figure shows the inbound message flow.



On startup, the adapter subscribes to all subjects listed in the inbound section of the configuration file. Wildcard subscriptions are supported. Inbound messages are received via client API callbacks. The messages are passed on to the Message Distributor.

The Message Distributor converts each Rendezvous message into a record targeting one or more data streams. The records are now ready to be published to Event Stream Processor, but they are not published immediately. Records are queued, then picked up by the SpPublisher object on separate threads, one thread for each record queue. You can configure the queue capacity. A larger queue is less likely to overflow in the event of a message burst. When the queue becomes three-quarters full, a warning is logged. Another warning is logged when the queue returns to three-quarters empty. If the queue is full, the adapter waits until room becomes available before placing the next record.

Records are published asynchronously. The adapter receives no feedback from Event Stream Processor. In the event of a failover, the Pub/Sub API switches, as configured to the spare Event Stream Processor instance without message loss.

Note: For outbound records, opcodes (values for ESP_OPS) are not communicated to the TIBCO Rendezvous server. For inbound records, all records have "p" (upsert) set as the opcode before publishing to the Server.

Datatype Mapping for the TIBCO Rendezvous Adapter

Event Stream Processor datatypes map to TIBCO Rendezvous datatypes.

Event Stream Processor Data-type	TIBCO Datatype
boolean	TibcorvMsg.Bool

Event Stream Processor Data-type	TIBCO Datatype
integer	TibcorvMsg.i32
long	TibcorvMsg.i64
float/money/money1-money15	TibcorvMsg.f64
string	TibcorvMsg.string
date/timestamp	TibcorvMsg.datetime
bigdatetime	TibcorvMsg.i64
interval	TibcorvMsg.i64
binary	TibcorvMsg.string

Setting the JAVA_HOME Environment Variable

Set the JAVA_HOME environment variable to point to the Java directory.

Prerequisites

- Install Java Runtime Environment version 1.6.0_26 or higher.
- Install the TIBCO Rendezvous binary libraries for the operating system on which the adapter is running. The binary libraries are not included in the TIBCO Rendezvous adapter distribution.

Task

Set the JAVA_HOME environment variable to the directory path where Java Runtime Environment 1.6.0_26 or higher is installed.

Next

- Place the TIBCO Rendezvous binary libraries in \$ESP_HOME/adapters/tibco_rv/lib/tibco/<platform_type> where <platform_type> is retrieved by the **arch** command for UNIX-based systems. For windows, win32 and win64 folders are present. Use these respective folders to copy the libraries.
- Verify that the ESP_HOME environment variable is set correctly.

Configuration

Configuration information for the TIBCO Rendezvous adapter.

TIBCO Rendezvous Adapter Directory

The adapter directory contains all files, such as configuration files, templates, examples, and JAR files, relating to the adapter.

```

README.txt Quick Guide
ReleaseNotes.txt Release Notes

bin/
  adapter.bat Standalone adapter startup script
  adapter.sh Standalone adapter startup script
  adapter-plugin.bat Plug-in connector startup script
  adapter-plugin.sh Plug-in connector startup script

config/
  controller.xsd Controller schema
  log4j.properties Sample logging configuration
  tibcorvadapter.xsd Adapter schema
  login.config Authentication configuration

example/ Working example
  GD Working example for guaranteed delivery

lib/
  tibco/
    i86pc
    sun4
    win32
    win64
    x84_64
  esp_tibco_rv_adapter.jar Tibco Rendezvous adapter library
  tibrvj.jar Tibco Rendezvous java library

javadoc/
  adapterapi/ Adapter API Javadoc
  tibcorvadapter/ TIBCO Rendezvous Adapter Javadoc

```

Common jars are located:

```

$ESP_HOME/adapters/jar

activation.jar Java mail library
adapterapi.jar Adapter API code
axis.jar Webservices jar
commons-codec-1.3.jar Required by SDK API
commons-discovery-0.2.jar
commons-logging-1.1.jar Logging library
esp_java_sdk-0.4.jar ESP SDK library
jaxrpc.jar Required by ESP SDK
log4j-1.2.14.jar Logging library
mail.jar Java mail library
saa.jar Webservices jar
ws-commons-util-1.0.2.jar Required by ESP SDK
wsdl4j-1.5.1.jar
xercesImpl.jar XML parser library

```

CHAPTER 2: Adapters Supported by Event Stream Processor

```
xmlrpc-client-3.1.3.jar Required by ESP SDK  
xmlrpc-common-3.1.3.jar Required by ESP SDK  
xmlrpc-server-3.1.3.jar Required by ESP SDK
```

Schema and Configuration File

The adapter configuration loads from a file and validates against the adapter schema.

The example folder contains a sample adapter configuration file.

Provide a valid configuration file, and ensure the adapter configuration specified in the file validates against the adapter schema.

Adapter Controller Parameters

The adapter **controllerPort** parameter specifies the adapter command and control port.

This parameter is defined in the `controller.xsd` file located in the `config` directory.

Parameter Name	Type	Description
controllerPort	positive integer	(Required) Specifies the adapter command and control port. User commands are sent to this port on localhost.

Event Stream Processor Parameters

Event Stream Processor parameters configure communication between Event Stream Processor and the TIBCO Rendezvous adapter.

These parameters are defined in the `controller.xsd` file in the `config` directory.

Parameter Name	Type	Description
espAuthType	string	(Required) Specifies method used to authenticate to the Event Stream Processor. Valid values are: <ul style="list-style-type: none">• server_rsa – RSA authentication using keystore• user_password – Kerberos and LDAP authentication• none – No authentication If the adapter is operated as a Studio plug-in, espAuthType is overridden by the Authentication Mode Studio start-up parameter.
espUser	string	(Required) Specifies user name required to log in to Event Stream Processor. It is required for any authentication scheme other than none (see espAuthType). No default value.

Parameter Name	Type	Description
espPassword	string	(Required) Specifies the password required to log in to Event Stream Processor. Required for any authentication scheme other than none (see espAuthType). Includes an "encrypted" attribute indicating whether the espPassword value is encrypted. Default value is false. If set to true, the password value is decrypted using espRSAKeyStore and espRSAKeyStorePassword .
espProjectUri	string	(Required) Specifies the total project Uri to connect to Event Stream Processor cluster. For example, <code>esp://localhost:19011/ws1/p1</code> .
pulseInterval	non-negative integer	(Optional) Specifies time interval, in seconds, during which outbound record changes are coalesced by Event Stream Processor, then received by the adapter as a single event. If not set or set to 0, record changes are received individually as they occur.
espHeartbeatPeriod	positive integer	(Optional) Specifies number, in seconds, that adapter waits before sending the next heartbeat to Event Stream Processor. If Event Stream Processor fails to receive two consecutive heartbeats, all records the adapter publishes are marked stale. Default value is 10.
recordQueueCapacity	positive integer	(Optional) Specifies capacity of the record queues. Default value is 4096.
maxPubPoolSize	positive integer	(Optional) Specifies the maximum size of the record pool. Record pooling allows for faster publication. Default value is 256.
maxPubPoolTime	positive integer	(Optional) Specifies 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.

Parameter Name	Type	Description
useTransactions	boolean	(Optional) If set to true, pooled messages are published to Event Stream Processor in transactions. If set to false, they are published in envelopes. Default value is false.
espRSAKeyStore	string	(Dependent required) Specifies the location of the RSA keystore, and is used to decrypt the password value. Required if espAuthType is set to <code>server_rsa</code> , or the encrypted attribute for espPassword is set to true, or both.
espRSAKeyStorePassword	string	(Dependent required) Specifies the keystore password, and is used to decrypt the password value. Required if espAuthType is set to <code>server_rsa</code> , or the encrypted attribute for espPassword is set to true, or both.
espEncryptionAlgorithm	string	(Optional) Used when the encrypted attribute for espPassword is set to true. If left blank, RSA is used as default.

Stream Configuration

Use the streams section in the configuration file to map message subjects to data streams.

Input Stream Parameters

The **name** and **subjects** parameters defined within the **inboundStreamType** parameter specify the name of the data stream and message subjects.

These parameters are defined in the `tibcorvadapter.xsd` file located in the `config` folder.

Property ID	Type	Description
name	string	(Required) Specifies the name of the data stream.
subjects	subjectsType	(Required) Specifies 0 or more message subjects. On startup, the adapter subscribes to each of these subjects. Wildcard subscriptions are supported. If an inbound message arrives on any of the specified subjects, it is published to this data stream. Several data streams can specify the same subjects.

Output Stream Parameters

The output stream parameters are defined within the **outboundStreamType** parameter.

These parameters are defined in the `tibcorvadapter.xsd` file in the `config` folder.

Parameter ID	Type	Description
name	string	(Required) Specifies the name of the data stream.
gdmode	boolean	(Advanced) If set to true, the adapter runs in guaranteed delivery (GD) mode and all GD-related parameters become required. Default value is false.
gdkeycolumnname	string	(Advanced) Specifies column name in the Flex operator holding the GD key. The GD key is a constantly increasing value that uniquely identifies every event regardless of the opcode in the stream of interest. No default value.
gdopcodecolumn-name	string	(Advanced) Specifies name of column in Flex operator holding opcode. The opcode is the operation code (for example, inserts, update, or delete) of the event occurring in the stream of interest. No default value.
gdcontrolstream	string	(Advanced) Specifies name of the control window in the GD setup. The control window is a source stream that informs the Flex operator of which data has been processed by the adapter and can be safely deleted. No default value.
gdbatchsize	integer	(Advanced) Specifies number of records after which the control window must be updated with the latest GD key. Default value is 1000.
gdpurgeinterval	integer	(Advanced) Specifies number of records after which to purge the Flex operator. Default value is 1000.

See also

- *Chapter 5, Guaranteed Delivery* on page 543

Rendezvous Server Settings

Configure the Rendezvous Server settings using the **rvDaemon**, **rvService**, **rvNetwork**, **cmmode**, **cmname**, and **cmledgerfile** parameters.

These parameters are defined in the `tibcorvadapter.xsd` file in the `config` folder.

Parameter Name	Type	Description
rvDaemon	string	(Required) Specifies the colon-separated host name (or IP address) and port on which the Rendezvous server daemon runs.
rvService	string	(Optional) Specifies the name of the Rendezvous service.
rvNetwork	string	(Optional) Specifies the name of the Rendezvous network.
cmmode	boolean	(Required) Enables Certified Message (CM) transport mode for both input and output streams. Default value is false.
cmname	string	(Optional) Is a reusable name that identifies the CM transport to other CM transports. If its value is not specified in CM mode, then it defaults to 'sesp'. Its value is considered only if cmmode is set to true.
cmledgerfile	string	(Optional) Specify a valid file location to run the CM transport in file-based ledger mode. Otherwise, it runs in process-based ledger mode. Its value is considered only if cmmode is set to true.

Sample TIBCO Rendezvous Configuration File

Sample configuration file (`adapter.xml`) for the TIBCO Rendezvous adapter.

This file is in the `example` folder.

```
<adapter>

<!-- Adapter controller -->
<controller>
  <controllerPort>13579</controllerPort>
</controller>

<!-- Sybase Stream processor settings -->
<esp>
  <espConnection>
    <espProjectUri>esp://localhost:19011/w1/p1</espProjectUri>
  </espConnection>

  <espSecurity>
    <espUser>espuser</espUser>
    <espPassword encrypted="false">espuser</espPassword>
  </espSecurity>
</esp>
</adapter>
```

```

        <espAuthType>none</espAuthType>
<!--
        <espRSAKeyStore>/keystore/keystore.jks</espRSAKeyStore>
        <espRSAKeyStorePassword>Sybase123</espRSAKeyStorePassword> --
>
        <espEncryptionAlgorithm>RSA</espEncryptionAlgorithm>
        </espSecurity>
        <maxPubPoolSize>1</maxPubPoolSize>
</esp>

<!-- Stream to subject mapping -->
<streams>
  <inbound>
    <stream>
      <name>MyInStream</name>
      <subjects>
        <subject>MySubject</subject>
      </subjects>
    </stream>
  </inbound>
  <outbound>
    <stream>
      <name>MyOutStream</name>
      <gdmode>>false</gdmode>
      <gdkeycolumnname>gdkey</gdkeycolumnname>
      <gdopcodecolumnname>gdopcode</gdopcodecolumnname>
      <gdcontrolstream>W1_truncate</gdcontrolstream>
      <gdbatchsize>2</gdbatchsize>
      <gdpurgeinterval>4</gdpurgeinterval>
    </stream>
  </outbound>
</streams>

<!-- Rendezvous settings -->
<rvSettings>
  <rvDaemon>localhost:7500</rvDaemon>
  <cmmode>>false</cmmode>
  <cmname>sesp</cmname>
  <cmledgerfile>C:\ledger.txt</cmledgerfile>
</rvSettings>

</adapter>

```

TIBCO Rendezvous Adapter

The TIBCO Rendezvous adapter publishes stream data to and from a Rendezvous subject.

The authentication method is set to that of Event Stream Processor: none, rsa, or gssapi.

Install TIBCO Rendezvous Adapter version 1.0 or later to use this adapter.

Property Label	Property ID	Type	Description
Connector Directory Path	baseDir	directory	(Required) Specify the path to the adapter installation directory. This property is ignored if the Connector Remote Directory Path property is supplied. No default value.
Configuration File Path	configFilePath	configFilename	(Required) Specify the absolute path to the adapter configuration file. This property is ignored if the Remote Configuration File Path property is supplied. No default value.
Connector Remote Directory Path	remoteBaseDir	string	(Advanced) Specify the path to the adapter remote base directory (for remote execution only). If this property is supplied, the Connector Directory Path property is ignored. No default value.
Remote Configuration File Path	remoteConfigFilePath	string	(Advanced) Specify the absolute path to the adapter remote configuration file (for remote execution only). If this property is supplied, the Configuration File Path property is ignored. No default value.
PropertySet	propertyset	string	(Advanced) Specifies the name of the property set (a group of properties and values) you want to use from the project configuration file. If you specify the same properties in the project configuration file and the ATTACH ADAPTER statement, the values in the property set override the values defined in the ATTACH ADAPTER statement. No default value.

Logging

The TIBCO Rendezvous adapter uses the Apache log4j API to log errors, warnings, and information and debugging messages.

The `log4j.properties` file contains the logging configuration. A sample `log4j.properties` file is part of the adapter distribution.

Setting the logging level to `DEBUG` may result in very large log files. The default level is `INFO`. Refer to <http://logging.apache.org/log4j> for details on logging configuration.

Operation

Operate the adapter from the command line.

Ensure the `rvDaemon` and, optionally, `rvService` and `rvNetwork` parameters in the configuration are consistent with the Rendezvous server settings.

Ensure that the project that processes events contains inbound and outbound streams. Set the desired logging level in the `log4j.properties` file.

Starting the TIBCO Rendezvous Adapter

To start the TIBCO adapter from the command line, start Event Stream Processor and execute the `start` command.

1. Start Event Stream Processor.

Windows:

1. Start the example cluster.

```
cd %ESP_HOME%\cluster\nodes\node1
%ESP_HOME%\bin\esp_server --cluster-node node1.xml
```

2. Compile CCL to create CCX.

```
%ESP_HOME%\bin\esp_compiler -i model.ccl -o model.ccx
```

3. Deploy the project on the cluster.

```
%ESP_HOME%\bin\esp_cluster_admin" --uri=esp://localhost:19011
--username=sybase --password=sybase --add_project --workspace-
name=w1 --project-name=p1 --ccx=model.ccx
```

4. Start the deployed project on the cluster.

```
%ESP_HOME%\bin\esp_cluster_admin" --uri=esp://localhost:19011
--username=sybase --password=sybase --start_project --
workspace-name=w1 --project-name=p1
```

UNIX:

1. Start the example cluster.

```
cd $ESP_HOME/cluster/nodes/node1
$ESP_HOME/bin/esp_server --cluster-node node1.xml
```

2. Compile CCL to create CCX.

```
$ESP_HOME/bin/esp_compiler -i model.ccl -o model.ccx
```

3. Deploy the project on the cluster.

```
$ESP_HOME/bin/esp_cluster_admin" --uri=esp://localhost:19011
--username=sybase --password=sybase --add_project --workspace-
name=w1 --project-name=p1 --ccx=model.ccx
```

4. Start the deployed project on the cluster.

```
$ESP_HOME/bin/esp_cluster_admin" --uri=esp://localhost:19011
--username=sybase --password=sybase --start_project --
workspace-name=w1 --project-name=p1
```

2. Start the adapter.

Operating System	Step
UNIX	Open a terminal window and enter: <pre>cd \$ESP_HOME/adapters/tibco_rv/bin ./adapter.sh <configuration file path> start</pre>
Windows	Open a command window and enter: <pre>cd %ESP_HOME%\adapters\tibco_rv\bin adapter.bat <configuration file path> start</pre>

Note: Use the **esp_subscribe** utility to ensure that inbound Rendezvous messages are successfully published to Event Stream Processor.

See also

- *Start Command* on page 489

Checking the TIBCO Rendezvous Adapter Status

To check the TIBCO adapter status from the command line, execute the **status** command.

Operating System	Step
UNIX	Open a terminal window and enter: <pre>cd \$ESP_HOME/adapters/tibco_rv/bin ./adapter.sh <configuration file path> status</pre>
Windows	Open a command window and enter: <pre>cd %ESP_HOME%/adapters/tibco_rv/bin adapter.bat <configuration file path> status</pre>

You see the adapter status, which is either running or stopped.

See also

- *Status Command* on page 490

Stopping the TIBCO Rendezvous Adapter

To stop the TIBCO adapter from the command line, execute the **stop** command.

Operating System	Step
UNIX	Open a terminal window and enter: <pre>cd \$ESP_HOME/adapters/tibco_rv/bin ./adapter.sh <configuration file path> stop</pre>
Windows	Open a command window and enter: <pre>cd %ESP_HOME%/adapters/tibco_rv/bin adapter.bat <configuration file path> stop</pre>

See also

- *Stop Command* on page 489

Example: Subscribing and Publishing

Subscribe to a subject, upload an outbound record on that subject to Event Stream Processor, send the message to the Rendezvous server, and then receive and publish it as an inbound record.

Prerequisites

You have a network connection to a running instance of the TIBCO Rendezvous server. Operate this example from the command line.

Task

1. Edit the `adapter.sh` script.
2. Set the `JAVA_HOME` environment variable to the directory where the Java Runtime Environment (JRE) is installed.
3. Start Event Stream Processor.

Operating System	Step
UNIX	Open a terminal window: <ol style="list-style-type: none"> 1. Start the example cluster: <code>start_server_cluster.sh</code> 2. Start the project on the cluster: <code>start_project.sh</code>
Windows	Open a command window: <ol style="list-style-type: none"> 1. Start the example cluster: <code>start_server_cluster.bat</code> 2. Add project to the cluster, and start it on the cluster: <code>start_project.bat</code>

4. Start the adapter.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./adapter.sh</code>
Windows	Open a command window and enter: <code>adapter.bat</code>

5. Wait five to ten seconds for the adapter to initialize.
6. Upload the outbound record.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./upload.sh</code>
Windows	Open a command window and enter: <code>upload.bat</code>

7. Start the Event Stream Processor subscriber utility to view data stream content.

Operating System	Step
UNIX	Open a terminal window and enter: <code>./esp-subscribe.sh</code>
Windows	Open a command window and enter: <code>esp-subscribe.bat</code>

8. Note the inbound record published to Event Stream Processor.

Use the Event Stream Processor internal adapter framework to build custom internal adapters, and ESP-supplied SDKs to build custom external adapters.

See also

- *Adapter Parameters Datatypes* on page 8

Custom Internal Adapters

For circumstances where the supplied adapters do not meet your needs, Event Stream Processor provides an internal adapter framework you can use to build internal adapters.

To create a custom internal adapter, provide a custom adapter library. Your custom adapter library uses the Event Stream Processor shared utility library to help convert external data to server format.

The adapter shared utility library exposes APIs with a C interface that allows you to implement life cycle and information management functions for your custom adapter. The C interface lets you to implement your custom adapter in C or C++ without any compiler restrictions.

The header file, `GenericAdapterInterface.h`, contains import declarations for functions in the adapter shared utility library.

See also

- *Custom Adapters* on page 3
- *Custom External Adapters* on page 517

The Adapter Shared Utility Library

The adapter shared utility library provides the utility functions required for a custom adapter implementation, including data conversion, data manipulation, and data management.

The header file, `GenericAdapterInterface.h`, contains the import declarations required to call utility functions in the adapter shared utility library.

When calling functions, each data utility requires a unique handle. The adapter shared utility library is labeled `esp_adapter_util_lib.dll` for Windows installations, and `libesp_adapter_util_lib.so` for Linux installations.

When calling functions in the adapter shared utility shared, each data utility requires a unique handle. For example, users can use the **ConnectionRow** function by calling

CreateConnectionRow. This call returns a unique handle in the form of a void pointer. The user can pass this pointer back when making calls to any other APIs under **ConnectionRow.**

Callback Functionality

An adapter implementation can use callback functions to log on to the Server, retrieve information related to schemas, and convey state information to the Server.

Callback functions are contained in the `esp_server_lib.dll` file for Windows installations, and in the `libesp_server_lib.dll` file for Linux installations.

The `GenericAdapterInterface.h` file contains import declarations for these functions.

Sample Model File

Sample syntax you can use to build a basic model file.

This model represents a schema with two columns of string data. The model also defines an input connection that references a sample custom adapter implementation.

```
CREATE MEMORY STORE "memory" PROPERTIES INDEXTYPE ='tree',
INDEXSIZEHINT =8;

CREATE INPUT WINDOW Custom
SCHEMA (column1 STRING, column2 STRING)
PRIMARY KEY (column1)
STORE "memory";

ATTACH INPUT ADAPTER Connection1
TYPE custom_in
TO Custom;
```

The Adapter Configuration File

The internal adapter configuration file is an XML file (`.cnxml`) that contains the properties and commands used by Event Stream Processor to start and stop the internal adapter, as well as other information that allows the internal adapter to be configured from the Studio.

The adapter configuration file also constructs the name for your custom adapter DLL file. The library name is referenced when you load your adapter.

This is sample code for naming the custom adapter DLL file:

```
<?xml version="1.0" encoding="UTF-8"?>
<Adapter type="input"
id="custom_in"
label="Custom Input"
descr="Dummy Custom Input Adapter"
>
<Library file="custom_in" type=binary"/>
<Section></Section>
</Adapter>
```

Note: The `.cnxml` file adheres to the `Adapter.xsd` file.

Once the `.cnxml` and custom adapter DLL files are ready, copy them to the `ESP_HOME/lib/adapters` folder.

Adapter Life Cycle Functions

All adapters follow a set of adapter life cycle events.

API	Description
bool reset(void* adapters);	This is the first life cycle API. Call this API to initialize both input and output adapters.
void start(void* adapter);	Call this API immediately after reset and use it for data processing specific to the adapter implementation. Call this API for both input and output adapters.
void* getNext(void* adapter);	This API reads data and returns a pointer to the data in a format the server understands. The adapter shared utility library provides data conversion functions. Call this API for input adapters only.
void putNext(void* adapter, void* stream);	This API converts presented data into a format the server understands, and writes it according to the specified adapter implementation. The adapter shared utility library provides data conversion functions. Call this API for output adapters only.
void stop(void* adapter);	Call this API to stop an adapter. This API gets called for both input and output adapters.
void cleanup(void* adapter);	Call this API to perform clean-up activities after an adapter is stopped.
void commitTransaction(void* adapter);	This API notifies an output adapter when a transaction ends. Call this API for output adapters only.
void putStartSync(void* adapter);	This API notifies the adapter implementation that the base data is being sent. Call this API for output adapters only.
void putEndSync(void* adapter);	This API notifies the adapter implementation that the base data has been sent. Call this API for output adapters only.

API	Description
<code>void purgePending(void* adapter);</code>	This API instructs an output adapter to purge pending data. Call this API for output adapters only.
<code>bool isOutBase(void* adapter);</code>	Call this API to determine whether the adapter expects to receive the output for the base contents of the stream. Call this API for output adapters only.

Adapter Setup Functions

The Server uses information management functions to complete the adapter implementation process.

These functions are used for both input and output adapters.

API	Description
<code>void* createAdapter();</code>	This is the first call the Server makes when creating an adapter. The function returns a unique handle that the Server uses to make subsequent calls to the adapter.
<code>void setCallBackReference(void*adapter,void* connectionCallBackReference);</code>	The Server calls this API to give the adapter implementation a unique handle that corresponds to the connectionCallBack object on the Server side. Use this handle as a parameter when making callbacks to the Server.
<code>void setConnectionRowType(void* adapter,void* connectionRowType);</code>	The Server calls this API to provide the adapter implementation with information related to schema.
<code>void setConnectionParams(void* adapter, void* connectionParams);</code>	The Server calls this API to provide the adapter implementation with information related to connection parameters.

Miscellaneous Functions

Describes miscellaneous APIs supported for Event Stream Processor adapters.

API	Description
<code>int getNumGood(void* adapter);</code>	The server calls this API to retrieve information from the adapter implementation about the number of good rows processed by the adapter.

API	Description
int getNumBad(void* adapter);	The server calls this API to retrieve information from the adapter implementation about the number of bad rows processed by the adapter.
int getNumRows(void* adapter);	The server calls this API to get information from the adapter implementation about the total number of rows processed by the adapter.
bool canDiscover(void* adapter);	The server calls this API to retrieve information from the adapter implementation about whether it supports schema discovery functionality.
bool hasError(void* adapter);	The server calls this API to get information from the adapter implementation about whether there were any errors during the processing of data.
void getError(void* adapter, char**errorString);	The server calls this API to get error information from the adapter implementation.

Adapter Run States

Adapters progress through a set of run states (RS) as they interact with Event Stream Processor.

- **RS_READY** – indicates that adapters are ready to be started.
- **RS_INITIAL** – indicates that the adapter is performing start-up and initial loading. An adapter enters the **RS_INITIAL** state when the reset function is called.
- **RS_CONTINUOUS** – indicates that the adapter is continuously waiting for additional data. If the **RS_CONTINUOUS** state is set in the reset method, input adapters return from reset with data to process, and output adapters return from reset prepared to accept 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. The adapter does not operate until the restart function is called.

When polling is enabled, an input adapter may change states from **RS_CONTINUOUS** to **RS_IDLE**. Change the adapter state back to **RS_CONTINUOUS** to retry data retrieval after a certain amount of time.

Schema Discovery for Internal Custom Adapters

Use extern "C" functions to enable your custom internal adapter for schema discovery.

Method	Description
extern "C" DLLEXPORT bool canDiscover(void* adapter)	During discovery, this is the first method called by the adapter framework to check whether an adapter supports discovery. Adapters that support discovery return a value of true.
extern "C" DLLEXPORT void setDiscovery(void* adapter)	This method tells the adapter that it is running in discovery mode.
extern "C" DLLEXPORT int getTableNames(void* adapter, char*** tables)	This method is a pointer to an array of strings that populates tables. The method contains table names when discovering tables in a database, or file names when discovering particular types of files in a directory.
extern "C" DLLEXPORT int getFieldNames(void* adapter, char*** names, const char* tableName)	This method returns field names.
extern "C" DLLEXPORT int getFieldTypes(void* adapter, char*** types, const char* tableName)	This method returns field types.
extern "C" DLLEXPORT int getSampleRow(void* adapter, char*** row, const char* tableName, int pos)	This method returns sample rows.

Sample Custom Internal Adapter Implementation

Sample syntax you can use to build your custom internal adapter implementation. This implementation incorporates extern "C" methods that enable schema discovery in a custom adapter.

```

/*
 * CustomAdapterInterface.cpp
 *
 * Author: sample
 */

#include "GenericAdapterInterface.h"
#include <vector>
#include <sstream>
#include <iostream>
#include <string>

using namespace std;

```



```

struct InputAdapter
{
    InputAdapter();
    void* connectionCallbackReference;
    void* schemaInfomartion;
    void* parameters;
    void* rowBuf;
    void* errorObjIdentifier;
    int _badRows;
    int _goodRows;
    int _totalRows;
    int getColumnCount();
    void setState(int st);
    bool discoverTables();
    bool discover(string tableName);
    vector<string> _discoveredTableNames;
    vector<string> _discoveredFieldNames;
    vector<string> _discoveredFieldTypes;
    vector<vector<string> > _discoveredRows;
    vector<string> _row1;
    vector<string> _row2;
    bool _discoveryMode;
};

InputAdapter::InputAdapter()
{
    rowBuf = NULL;
    _badRows = 0;
    _goodRows = 0;
    _totalRows = 0;
    _discoveryMode = false;
    _discoveredTableNames.clear();
    _discoveredFieldNames.clear();
    _discoveredFieldTypes.clear();
    _discoveredRows.clear();
    _row1.clear();
    _row2.clear();
}

int InputAdapter::getColumnCount()
{
    return ::getColumnCount(schemaInfomartion);
}

void InputAdapter::setState(int st)
{
    ::setAdapterState(connectionCallbackReference, st);
}

extern "C" DLLEXPORT
void* createAdapter()
{
    return new InputAdapter();
}

```

CHAPTER 3: Custom Adapters

```
extern "C" DLLEXPORT
void setCallbackReference(void *adapter,void
*connectionCallbackReference)
{
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;
    inputAdapterObject->connectionCallbackReference =
connectionCallbackReference;
}

extern "C" DLLEXPORT
void setConnectionRowType(void *adapter,void *connectionRowType)
{
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;
    inputAdapterObject->schemaInfomartion = connectionRowType;
}

extern "C" DLLEXPORT
void setConnectionParams(void* adapter,void* connectionParams)
{
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;
    inputAdapterObject->parameters = connectionParams;
}

extern "C" DLLEXPORT
void* getNext(void *adapter)
{
    StreamRow streamRow = NULL;
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;
    int n = inputAdapterObject->getColumnCount();
    std::stringstream ss;
    if(inputAdapterObject->_totalRows <10){

        for (int column = 0; column < n; column++) {
            ss.str("");
            ss << inputAdapterObject->_totalRows;
            std::string tempString;
            tempString = ss.str();
            std::string row = "ROW";
            row.append(tempString);
            ss.str("");
            ss << column;
            tempString = ss.str();
            std::string columnString = "COLUMN";
            columnString.append(tempString);
            row.append(columnString);
            ::setFieldAsStringWithIndex(inputAdapterObject->rowBuf,
column, row.c_str());

        }

        inputAdapterObject->_totalRows++;
        streamRow = ::toRow(inputAdapterObject->rowBuf,
inputAdapterObject->_totalRows, inputAdapterObject-
>errorObjIdentifier);
        if( streamRow )
        {
```

```

        inputAdapterObject->_goodRows++;
    } else
    {
        inputAdapterObject->_badRows++;
    }

    } else {
        inputAdapterObject->setState(RS_DONE);
    }
    return streamRow;
}

extern "C" DLLEXPORT
bool reset(void *adapter)
{
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;
    if(inputAdapterObject->rowBuf)
        delete inputAdapterObject->rowBuf;
    string type = "RowByOrder";
    inputAdapterObject->rowBuf
= ::createConnectionRow(type.c_str());
    ::setStreamType(inputAdapterObject->rowBuf, inputAdapterObject-
>schemaInfomartion, false);
    inputAdapterObject->errorObjIdentifier
= ::createConnectionErrors();
    inputAdapterObject->setState(RS_CONTINUOUS);
    return true;
}

extern "C" DLLEXPORT
int getTotalRowsProcessed(void *adapter)
{
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;
    return inputAdapterObject->_totalRows;
}

extern "C" DLLEXPORT
int getNumberOfBadRows(void *adapter)
{
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;
    return inputAdapterObject->_badRows;
}

extern "C" DLLEXPORT
int getNumberOfGoodRows(void *adapter)
{
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;
    return inputAdapterObject->_goodRows;
}

extern "C" DLLEXPORT
bool hasError(void *adapter)
{
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;

```

```

    return ! (::empty(inputAdapterObject->errorObjIdentifier));
}

extern "C" DLLEXPORT
void getError(void *adapter, char** errorString)
{
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;
    ::getAdapterError(inputAdapterObject->errorObjIdentifier,
errorString);
}

extern "C" DLLEXPORT
void start(void* adapter){}

extern "C" DLLEXPORT
void stop(void* adapter){}

extern "C" DLLEXPORT
void cleanup(void* adapter){}

extern "C" DLLEXPORT
bool canDiscover(void* adapter){return true;}

extern "C" DLLEXPORT
void deleteAdapter(void* adapter)
{
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;
    delete inputAdapterObject;
}

extern "C" DLLEXPORT
void commitTransaction(void *adapter){}

extern "C" DLLEXPORT
int getTableNames(void* adapter, char*** tables)
{
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;

    if(!inputAdapterObject->discoverTables())
    {
        return 0;
    }

    (*tables) = (char**) malloc(sizeof(char*)*inputAdapterObject-
>_discoveredTableNames.size());

    for(int index=0; index < inputAdapterObject-
>_discoveredTableNames.size(); index++)
    {
        size_t tableNameSize = inputAdapterObject-
>_discoveredTableNames[index].size() + 1 ;
        char* tableName = new char [tableNameSize ];
        strncpy(tableName, inputAdapterObject-
>_discoveredTableNames[index].c_str(),tableNameSize);
    }
}

```

```

        (*tables)[index] = tableName;
    }

    return inputAdapterObject->_discoveredTableNames.size();
}

extern "C" DLLEXPORT
int getFieldNames(void* adapter, char*** names, const char*
tableName)
{
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;

    string table (tableName);

    if(!inputAdapterObject->discover(table))
    {
        return 0;
    }

    (*names) = (char**) malloc(sizeof(char*)*inputAdapterObject-
>_discoveredFieldNames.size());

    for(int index=0; index < inputAdapterObject-
>_discoveredFieldNames.size(); index++)
    {
        size_t fieldNameSize = inputAdapterObject-
>_discoveredFieldNames[index].size() + 1;
        char* fieldName = new char [ fieldNameSize ];
        strncpy(fieldName, inputAdapterObject-
>_discoveredFieldNames[index].c_str(),fieldNameSize);
        (*names)[index] = fieldName;
    }

    return inputAdapterObject->_discoveredFieldNames.size();
}

extern "C" DLLEXPORT
int getFieldTypes(void* adapter, char*** types, const char*
tableName)
{
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;

    string table (tableName);

    if(!inputAdapterObject->discover(table))
    {
        return 0;
    }

    (*types) = (char**) malloc(sizeof(char*)*inputAdapterObject-
>_discoveredFieldTypes.size());

    for(int index=0; index < inputAdapterObject-
>_discoveredFieldTypes.size(); index++)

```

```

    {
        size_t fieldTypeSize = inputAdapterObject-
>_discoveredFieldTypes[index].size() + 1;
        char* fieldType = new char [ fieldTypeSize ];
        strncpy(fieldType, inputAdapterObject-
>_discoveredFieldTypes[index].c_str(), fieldTypeSize);
        (*types)[index] = fieldType;
    }

    return inputAdapterObject->_discoveredFieldTypes.size();
}

extern "C" DLLEXPORT
int getSampleRow(void* adapter, char*** row, const char* tableName,
int pos)
{
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;

    string table (tableName);

    if(!inputAdapterObject->discover(table))
    {
        return 0;
    }

    vector<string> vals;

    if (pos < (int)inputAdapterObject->_discoveredRows.size())
    {
        vals = inputAdapterObject->_discoveredRows[pos];

        (*row) = (char**) malloc(sizeof(char*)*vals.size());

        for(int index=0; index < vals.size(); index++)
        {
            size_t columnSize = vals[index].size() + 1;
            char* column = new char [ columnSize ];
            strncpy(column, vals[index].c_str(),columnSize);
            (*row)[index] = column;
        }
    }

    return vals.size();
}

extern "C" DLLEXPORT
void setDiscovery(void* adapter)
{
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;
    inputAdapterObject->_discoveryMode = true;
}

bool InputAdapter::discoverTables()
{
    _discoveredTableNames.push_back("Table1");
}

```

```

        _discoveredTableNames.push_back("Table2");
        _discoveredTableNames.push_back("Table3");
        _discoveredTableNames.push_back("Table4");
        _discoveredTableNames.push_back("Table5");
        return true;
    }

bool InputAdapter::discover(string tableName)
{
    _discoveredFieldNames.clear();
    _discoveredFieldTypes.clear();
    _row1.clear();
    _row2.clear();
    _discoveredRows.clear();
    _discoveredFieldNames.push_back("Column1");
    _discoveredFieldNames.push_back("Column2");
    _discoveredFieldNames.push_back("Column3");
    _discoveredFieldNames.push_back("Column4");
    _discoveredFieldNames.push_back("Column5");
    _discoveredFieldTypes.push_back("integer");
    _discoveredFieldTypes.push_back("string");
    _discoveredFieldTypes.push_back("string");
    _discoveredFieldTypes.push_back("float");
    _discoveredFieldTypes.push_back("float");
    _row1.push_back("1");
    _row1.push_back("A");
    _row1.push_back("B");
    _row1.push_back("1.1");
    _row1.push_back("2.2");
    _row2.push_back("2");
    _row2.push_back("X");
    _row2.push_back("Y");
    _row2.push_back("3.3");
    _row2.push_back("4.4");
    _discoveredRows.push_back(_row1);
    _discoveredRows.push_back(_row2);
    return true;
}

```

Custom External Adapters

The external adapter framework provides a mechanism to create and add adapters that are not included in the set provided by Event Stream Processor.

External adapters can be added in the field and provide all necessary information to allow the Studio and Server to manage and interact with an external datasource or sink. These custom adapters are simply datasource and sink applications that are built with the Pub/Sub API. After an adapter configuration file (cnxml) is installed, an external adapter can be used in the Studio in the same way as built-in adapters.

See also

- *Custom Adapters* on page 3

- *Custom Internal Adapters* on page 505
- *External Adapters* on page 127

External Adapter Configuration File

The framework defines the structure of the external adapter configuration file (cnxml) file that contains the adapter properties.

The external adapter configuration file is an XML file that contains the properties and commands used by Event Stream Processor to start and stop the external adapter and to optionally run schema discovery, as well as other information that allows the external adapter to be configured from Studio.

The example below shows a cnxml file that uses four of the utilities shipped with Event Stream Processor (**esp_convert**, **esp_upload**, **esp_client**, and **esp_discxmlfiles**) to fully define a functional external adapter that supports browsing a directory of files, the creation of a source stream, and data loading. This sample configuration file, `simplified_xml_input_plugin.cnxml`, can be found in the `$ESP_HOME/lib/adapters` directory. The directory is included in the standard Event Stream Processor distribution package.

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

```
<?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
    which are prefixed with "x_". Although these are parameters,
    the framwork requires them to be defined using the <Internal
    .../> element. They are hidden from the user in ESP 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 ESP Binary format using
esp_convert;
  pipe into the esp_upload program, naming the upload
connection:
  $platformStream.$platformConnection
-->
<Internal id="x_unixCmdExec"
  label="Execute Command"
  type="string"
  default="$ESP_HOME/bin/esp_convert -p $platformCommandPort
&lt;&quot;$directory/$filename&quot;; | $ESP_HOME/bin/esp_upload -m
$platformStream.$platformConnection -p $platformCommandPort"
/>
<Internal id="x_winCmdExec"
  label="Execute Command"
  type="string"
  default="$+/{$ESP_HOME/bin/esp_convert} -p $platformCommandPort
&lt;&quot;$directory/$filename&quot;; | $+/{$ESP_HOME/bin/esp_upload}
-m $platformStream.$platformConnection -p $platformCommandPort"
/>

<!--
  use the esp_client command to stop an existing esp_upload
connection named:
  $platformStream.$platformConnection
-->
<Internal id="x_unixCmdStop"
  label="Stop Command"
  type="string"
  default="$ESP_HOME/bin/esp_client -p $platformCommandPort 'kill
every {$platformStream.$platformConnection}' &lt;/dev/null"
/>
<Internal id="x_winCmdStop"
  label="Stop Command"
  type="string"
  default="$+/{$ESP_HOME/bin/esp_client} -p $platformCommandPort
&quot;kill every {$platformStream.$platformConnection}&quot;
&lt;nul"
/>

<!--
  Use the esp_discxmlfiles command to do data discovery.
  The command below will have '-o "<temp file>"' added to it. It
  will write the discovered data in this file.
-->
<Internal id="x_unixCmdDisc"
  label="Discovery Command"

```

```

        type="string"
        default="$ESP_HOME/bin/esp_discxmlfiles -d &quot;
$directory&quot;;"
    />
    <Internal id="x_winCmdDisc"
        label="Discovery Command"
        type="string"
        default="$+/{$ESP_HOME/bin/esp_discxmlfiles} -d &quot;$/
{$directory}&quot;;"
    />
</Special>

<Section>

    <!--
        Any parameter defined here, is visible in the ESP Studio, and
may
        be configured by the user at runtime in the data location
explorer.
        These are defined according to the $ESP_HOME/etc/Adapter.xsd
        schema.
    -->

    <Parameter id="filename"
        label="File"
        descr="File to upload"
        type="tables"
        use="required"
    />
    <Parameter id="directory"
        label="path to file"
        descr="directory to search"
        type="directory"
        use="required"
    />
    <Parameter id="propertyset"
        label="propertyset"
        descr="to look up properties in project configuration"
        type="string"
        use="advanced"
        default=""/>
</Section>
</Adapter>

```

External Adapter Properties

See examples in `$ESP_HOME/lib/connections/PLUGIN_TEMPLATE.cnxml` for a sample cnxml file that may be copied and customized. It has all possible internal parameters embedded in it, and has comment blocks indicating their usage.

Property Id	Type	Description
x_paramFile	string	Specifies the file name that where the adapter framework writes all internal and user-defined parameters. It may use other internal parameters in specifying the file name. For example: <code>/tmp/mymodel.\$platformStream.\$platformConnector.\$platformCommandPort.cfg</code>
x_paramFormat	string	Set to <code>prop</code> , <code>shell</code> , or <code>xml</code> to choose the format for the parameter file.
x_addParamFile	boolean	Determines if the parameter file name is automatically appended to all <code>x_cmd*</code> strings. For example, if you specify the command as <code>cmd -f</code> , and this is set to true, the actual command is executed as <code>cmd -f <value of x_paramFile></code> .
x_initialOnly	boolean	If true, does initial loading only. Set to false for continuous loading. Initial loading is useful for adapters that start, load some static data then finish, thus allowing another adapter group to start up in a staged loading scenario.
x_killRetryPeriod	integer	If this parameter is <code>>0</code> 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 it is equal to zero, run the x_{unix,win}CmdStop command only once and assume that it has stopped the x_{unix,win}CmdExec command.

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.

The discovery commands, **x_unixDiscCmd** and **x_winDiscCmd** always run on the Studio host. All other commands run on the Server host.

CHAPTER 3: Custom Adapters

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, it implies that the discovery command and the discovery file name that the framework generates are running and are generated in a Windows environment. The path to the discovery file is a Windows-specific path with drive letters and '\ ' 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.

Command	Description
x_unixCmdConfig x_winCmdConfig	The configure command should do any required parsing and/or checking of the parameters. It may also convert the parameters into the real format expected by the execution command by reading, parsing, and re-writing the parameter file. If the configure command fails (non-zero return), it is reported as a <code>reset ()</code> error, and the adapter fails to start.
x_unixCmdExec x_winCmdExec	When the Server starts the adapter, it executes this command with its ending indicating that the connector has finished.
x_unixCmdStop x_winCmdStop	The stop command runs from a separate thread; it should stop all processes created with the <code>x_{unix,win}CmdExec</code> command, thus causing the <code>x_{unix,win}CmdExec</code> to return.
x_unixCmdClean x_winCmdClean	The clean command runs after the Server has stopped the connection, that is, when <code>x_{unix,win}CmdExec</code> returns.

Command	Description
x_winDiscCmd	<p>This command is for schema discovery. It should write a discovery file into the file name passed to it. The parameter -o <temporary disc filename> argument is appended to this command before it is executed.</p> <pre data-bbox="552 357 1170 944"> <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> </pre>

User-Defined Parameters and Parameter Substitution

Internal parameters and any number of user-defined parameters can be created in the cxml 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. The simplest example is from the `simplified_xml_input_external.cxml` file. Some of the long lines below have been split for readability and to avoid formatting issues.

```

<Internal id="x_unixCmdExec"
  label="Execute Command"
  type="string"
  default="$ESP_HOME/bin/esp_convert
    -p $platformCommandPort &lt;&quot;$directory/
$filename&quot;; | $ESP_HOME/bin/esp_upload
    -m $platformStream.$platformConnection
    -p $platformCommandPort"
/>

```

CHAPTER 3: Custom Adapters

External environment variables, such as `ESP_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 then on the second pass `\\` turns into a single backslash.

Special substitution syntax for Windows convenience:

<code>\$/ {value}</code> <code>\$/+ {value}</code>	Replaces all the forward slashes in the value by backslashes, for convenience of specifying the Windows paths that otherwise would have to have all the slashes escaped.
<code>\$% {value}</code> <code>\$/% {value}</code>	Replaces all the <code>%</code> with <code>%%</code> as escaping for Windows.

If the resulting string is passed to shell or `cmd . exe` for execution, shell or `cmd . exe` would do its own substitution too.

Here is an example using 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.

```

<Internal id="x_unixCmdExec"
  label="Execute Command"
  type="string"
  default="$ESP_HOME/bin/esp_convert
    ${platformSsl==1?-e}
    ${dateFormat}<>-m '$dateFormat' }
    -c '${user=?user:$user}:$password'
    -p $platformCommandPort
    <"$directory/$filename" |
      $ESP_HOME/bin/esp_upload
      ${platformSsl==1?-e}
    -m $platformStream.$platformConnection
      -c '$user:$password'
    -p $platformCommandPort"
/>

```

Auto-Generated 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 either:

- Java properties
- Shell assignments
- Simple XML format

Commands then have full access to the parameter file.

There is an example of parameters in the `simplified_xml_input_plugin.cnxml` file.

```

    <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"
    default="false"
  />

```

The parameter file is written to `/tmp/PARAMETER_FILE.txt`.

```

directory=/home/sjk/work/aleri/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
```

or a full list of parameters, in the Java properties format. Note the format can be specified as `shell` for shell assignments, or as `xml` for a simple XML format.

When `x_addParamFile` is specified as `true`,

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

the argument `/tmp/PARAMETER_FILE.txt` is added to all commands prior to being executed.

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 on the read-only name brings up a different dialog, allowing you to modify file contents. This provides you with a way to specify user-editable configuration files.

Custom External Parameter Datatypes

The `adapter.xsd` schema supports several datatypes for user-defined parameters.

For supported datatypes refer to *Adapter Parameters Datatypes* in the *Introduction*.

Custom external adapters do not support the datatypes:

- `runtimeFilename`
- `runtimeDirectory`
- `text`
- `query`
- `permutation`

Note: The **start** and **stop** commands are run by the Server, while discovery is run by Studio. This distinction can affect use of these parameters.

See also

- *Adapter Parameters Datatypes* on page 8

Creating Custom External Adapters

General steps for using SDKs to build custom adapters.

1. Get an SDK instance.
2. Create credentials for the required type of authentication.
3. Connect to a project using those credentials.
4. Create a publisher to publish to the Server.
5. Create a subscriber to subscribe to records from the Server.
6. Publish or subscribe.

See also

- *Custom Adapters* on page 3
- *Java External Adapters* on page 527
- *C/C++ External Adapters* on page 531
- *.Net External Adapters* on page 534

Java External Adapters

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

See also

- *Custom Adapters* on page 3
- *C/C++ External Adapters* on page 531
- *.Net External Adapters* on page 534
- *Creating Custom External Adapters* on page 527

Connecting to a Project

Connect to a project using your authentication credentials.

1. Get the project:

```
String projectUriStr = "esp://localhost:19011/ws1/p1";
Uri uri = new Uri.Builder(projectUriStr).create();
project = sdk.getProject(uri, credentials);
```

2. Connect to the project:

```
project.connect(60000);
```

Here, 60000 refers to the time in milliseconds that the Server waits for the connection call to complete before timing out.

Creating a Publisher

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

CHAPTER 3: Custom Adapters

1. Create and connect to a publisher:

```
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, and publish:

```
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);
```

Sample Java Code for addRow

The addRow operation adds a single record row to messages published to the Server.

Opcodes are used 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 TIMESTAMP:   writer.setTimestamp((Date) attValue);
break;
} //switch
} //for loop
rowWriter.endRow();
```

Subscribing Using Callback

Perform callbacks for new data.

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 wish 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 which implements the `processEvent` method and gets called by the callback mechanism.

3. Create the callback class, which is used to register with the subscriber.

a) Implement `Callback<SubscriberEvent>`.b) Implement the `getName()` and `processEvent(SubscriberEvent)` methods.

```
public void processEvent(SubscriberEvent event) {
    switch (event.getType()) {
        case SYNC_START:    dataFromLogstore=true;    break;
        case SYNC_END:     dataFromLogStore=false;
        case ERROR:       handleError(event);
    break;
    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 is variable.

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`:

```
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; index = 0; j <
row.getSchema().getColumnCount(); ++j) {
            if ( row.isNull(j)) { record.add(index,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 TIMESTAMP: record.add(j,
row.getTimestamp(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.

Subscribe Using Direct Access Mode

Direct access mode is recommended only for testing purposes.

```

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

```

Publish Using Callback

Publishing in callback mode can be used in special cases, but is not recommended.

```

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 two methods: `getName()` and `processEvent(PublisherEvent event)`.

The script for implementing `processEvent` should look like this:

```

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;
                }
            } //columns
            mrowwriter.endRow();
        } //for
    }
}

```

```

        event.getPublisher().publish(mwriter);
        case ERROR: break;
        case DISCONNECTD:break;
    } //switch
} //processEvent

```

C/C++ External Adapters

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

See also

- *Custom Adapters* on page 3
- *Java External Adapters* on page 527
- *.Net External Adapters* on page 534
- *Creating Custom External Adapters* on page 527

Getting a Project

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

All calls to SDK are available as external C calls.

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);

```

Publishing and Subscribing

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

1. Create the publisher:

```

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:

Note: The sample code in this step includes syntax for adding rows to messages.

```
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;
}
} //for
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:

```
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 which will get called
up.
```

4. Subscribe using callback:

```

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:
            cleanupExit(); break;
        case ESP_SUBSCRIBER_EVENT_ERROR:
            handleError(event,error); break;
    }
} //end subscriber_callback

```

handleData

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

```

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;
        }
    }
}

```

.Net External Adapters

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

See also

- *Custom Adapters* on page 3
- *Java External Adapters* on page 527
- *C/C++ External Adapters* on page 531
- *Creating Custom External Adapters* on page 527

Connecting to the Server

Set credentials and .Net server options when you connect to the Server.

1. Run the **NetEspError** command to create an error message store for these tasks:

```
NetEspError error = new NetEspError();
```

2. Set a new URI:

```
NetEspUri uri = new NetEspUri();  
uri.set_uri("esp://cepsun64amd.sybase.com:19011", error);
```

3. Create your credentials:

```
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:

```
NetEspServerOptions options = new NetEspServerOptions();  
options.set_mode(NetEspServerOptions.NET_ESP_ACCESS_MODE_T.NET_CA  
LLBACK_ACCESS);  
server = new NetEspServer(uri, creds, options);  
int rc = server.connect(error);
```

Connecting to a Project

Use sample .Net code to connect to a project.

1. Get the project:

```
NetEspProject project = server.get_project("test", "test",  
error);
```

2. Connect to the project:

```
project.connect(error);
```

Publishing

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

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:

```
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:

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

Connecting to a Subscriber

Create and connect to a new subscriber.

1. Create a subscriber:

```
NetEspSubscriberOptions options = new NetEspSubscriberOptions();
options.set_mode(NetEspSubscriberOptions.NET_ESP_ACCESS_MODE_T.NE
T_CALLBACK_ACCESS);
NetEspSubscriber subscriber = new NetEspSubscriber(options,
error);
```

2. Connect to the subscriber:

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

Subscribing Using Callback Mode

Perform callbacks for new data.

1. Set the subscriber options:

```
NetEspSubscriberOptions options = new NetEspSubscriberOptions();
options.set_mode(NetEspSubscriberOptions.NET_ESP_ACCESS_MODE_T.NE
T_CALLBACK_ACCESS);
```

CHAPTER 3: Custom Adapters

```
NetEspSubscriber subscriber = new NetEspSubscriber(options,  
error);
```

2. Create the callback instance:

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

3. Create the callback registry:

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

4. Connect to the subscriber:

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

5. Subscribe to a stream:

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

6. Implement the callback:

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

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

```
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_DATE:
        dvalue = row_reader.get_date(i, error);
        Console.Write(dvalue + ", ");
        break;
    case
NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_TIMESTAMP:
        tvalue = row_reader.get_timestamp(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);
        Console.Write(System.Text.Encoding.Default.GetString(binvalue) +
", ");
        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.WriteLine(mon.get_long(error) + ", ");
        break;
    case
NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_MONEY10:
        mon = row_reader.get_money(i, error);
        Console.WriteLine(mon.get_long(error) + ", ");
        break;
    case
NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_MONEY15:
        mon = row_reader.get_money(i, error);
        Console.WriteLine(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.WriteLine(usecs + ", ");
        break;
    }
}
}

```

8. Disconnect from the subscriber:

```

        rc = subscriber.disconnect(error);
    }

```

CHAPTER 4 Schema Discovery

You can use the schema discovery feature to discover external schemas and create CCL schemas based on the format of the data from the datasource connected to an adapter.

Every row in a stream or window must have the same structure, or schema, which includes the column names, the column datatypes, and the order in which the columns appear. Multiple streams or windows may use the same schema, but a stream or window can only have one schema.

Rather than manually creating a new schema, you can use schema discovery to discover and automatically create a schema based on the format of the data from the datasource connected to your adapter. For example, for the Database Input adapter, you can discover a schema that corresponds to a specific table from a database the adapter is connected to.

To discover a schema, you need to first configure the adapter properties. Each adapter that supports schema discovery has unique properties that must be set to enable schema discovery.

Adapters that Support Schema Discovery

The adapters that support schema discovery and the properties they use to enable it.

Adapter	Supports Schema Discovery	Properties
AtomReader Input	No	—
Database Input	Yes	Database Service Name of database service from which the adapter obtains the database connection.
Database Output	Yes	Database Service Name of service entry to use.
File CSV Input	Yes	Directory The absolute path to the data files you want the adapter to read.
File CSV Output	No	—
File FIX Input	No	—
File FIX Output	No	—

Adapter	Supports Schema Discovery	Properties
File XML Input	Yes	Directory The absolute path to the data files you want the adapter to read.
File XML Output	No	—
FIX Input	No	—
FIX Output	No	—
Flex Output	No	—
HTTP Input	No	—
JMS CSV Input	Yes	<ul style="list-style-type: none"> • Delimiter – field delimiter • Connection Factory – connection factory class name • JNDI Context Factory – context factory for JNDI context initialization • JNDI URL • Destination Type • Destination Name
JMS CSV Output	No	—
JMS Custom Input	No	—
JMS Custom Output	No	—
JMS FIX Input	No	—
JMS FIX Output	No	—
JMS Object Array Input	Yes	<ul style="list-style-type: none"> • Connection Factory – connection factory class name • JNDI Context Factory – context factory for JNDI context initialization • JNDI URL • Destination Type • Destination Name
JMS Object Array Output	No	—

Adapter	Supports Schema Discovery	Properties
JMS XML Input	Yes	<ul style="list-style-type: none"> • Connection Factory – connection factory class name. • JNDI Context Factory – context factory for JNDI context initialization • JNDI URL • Destination Type • Destination Name
JMS XML Output	No	—
Kdb Input	Yes	<ul style="list-style-type: none"> • KDB Server • KDB Port • KDB User • KDB Password
Kdb Output	Yes	<ul style="list-style-type: none"> • KDB Server • KDB Port • KDB User • KDB Password
Log File Input	No	—
Random Tuples Generator Input	No	—
Replication Server Input	Yes	<ul style="list-style-type: none"> • RSSD Host • RSSD Port • RSSD Database Name • RSSD User Name • RSSD Password
Reuters Marketfeed Input	Yes	Discovery Path
Reuters Marketfeed Output	No	—
Reuters OMM Input	Yes	Discovery Path
Reuters OMM Output	No	—
RTView Output	No	—

Adapter	Supports Schema Discovery	Properties
SMTP Output	No	—
Socket (as Client) CSV Input	No	—
Socket (as Client) CSV Output	No	—
Socket (as Client) XML Input	No	—
Socket (as Client) XML Output	No	—
Socket (as Server) XML Input	No	—
Socket (as Server) XML Output	No	—
Socket (as Server) CSV Input	No	—
Socket (as Server) CSV Output	No	—
Socket FIX Input	No	—
Socket FIX Output	No	—
Sybase IQ Output	No	—
Open Input and Output	No	—
Tibco Rendezvous Input	No	—
Tibco Rendezvous Output	No	—
NYSE Input	Yes	Discovery Directory Path Absolute path to the adapter discovery directory.

See also

- *Internal Adapters* on page 18
- *External Adapters* on page 127

CHAPTER 5 **Guaranteed Delivery**

Guaranteed delivery (GD) is a delivery mechanism that guarantees data is processed from a stream to an adapter.

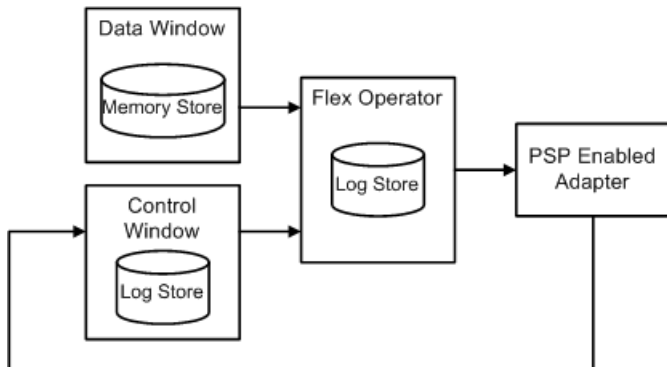
GD ensures that data continues to be processed when:

- The Server fails.
- The destination (third-party server) fails.
- The destination (third-party server) does not respond for a period of time.

Persistent subscribe pattern (PSP) is used to implement GD in output adapters. Input adapters support GD using facilities provided by the source rather than using PSP. The WebSphereMQ Input and Output adapter, all JMS Input and Output adapters, and the TIBCO Rendezvous adapter all support GD. These adapters have specific PSP and GD parameters that are unique to them. Examples for enabling GD in one of the JMS CSV Output adapters and the WebSphere Output adapter are located `<ESP_HOME>/examples/ccl/JmsOutBoundAdapterWithGDSupport` and `<ESP_HOME>/examples/ccl/WsmqOutBoundAdapterWithGDSupport` respectively.

PSP works through a combination of a window (input, output), a control window, and a Flex operator with a log store. The window and control window plug into the Flex operator. Data from the window on which PSP is enabled is entered into the flex operator, which generates a sequence number and opcode from the data, and places them at the beginning of each row of data. The Flex operator sends this data to the adapter that is attached to it, and the adapter passes the information on to the control window. Finally, the control window informs the Flex operator of the data that has been processed by the adapter, and the Flex operator removes this data from the log store.

Figure 12: PSP Overview



See also

- *TIBCO Rendezvous Adapter* on page 488
- *WebSphere MQ Input Adapter* on page 121
- *WebSphere MQ Output Adapter* on page 123
- *JMS CSV Input Adapter* on page 47
- *JMS CSV Output Adapter* on page 50
- *JMS Custom Input Adapter* on page 55
- *JMS Custom Output Adapter* on page 59
- *JMS FIX Input Adapter* on page 64
- *JMS FIX Output Adapter* on page 66
- *JMS Object Array Input Adapter* on page 70
- *JMS Object Array Output Adapter* on page 74
- *JMS XML Input Adapter* on page 79
- *JMS XML Output Adapter* on page 82
- *Output Stream Parameters* on page 497

Log Window

The log window is a Flex operator that is assigned to a log store and is the centre of the guaranteed delivery (GD) mechanism.

For persistent subscription using persistent subscribe pattern (PSP), attach the output adapter to a log window instead of a stream of interest. The stream definition for the log window contains all the columns belonging to the stream of interest, plus two additional columns. These additional columns are the `gdKey` (long) and the `gdOpcode` (integer).

The `gdKey` is a constantly increasing value that uniquely identifies every event, regardless of the opcode in the stream of interest. This serves as the key for the log window. The `gdOpcode` is the operation code (for example, INSERT, UPDATE, or DELETE) of the event that occurs in the stream of interest.

The log window takes two inputs namely from the stream whose data needs to be delivered in a guaranteed fashion (stream of interest) and from the truncate window. The log window has a method associated with each input. The method associated with the stream of interest:

1. Increments the `gdKey` by 1, starting from 0, on every incoming event. On restart, it starts from the last generated sequence number by self inspecting the data it has previously output.
2. Determines the opcode of the incoming event.
3. Outputs the `gdKey` and the `gdOpcode` determined in the previous two steps, along with all the columns of the input event from the stream of interest.

The method associated with the truncate window is responsible for ensuring that the data in the log window does not grow indefinitely. Every time an event occurs on the truncate window,

this method deletes all events in the log window that has a `gdKey` less than or equal to the provided `gdKey`, and provided that the purge data flag is set to `true`.

Truncate Window

Output adapters use the truncate window to inform the log window of which data has been processed by the adapter and can be safely deleted.

It has three columns, `simpleKey` (integer), `gdKey` (long), and `purge` (boolean). The `simpleKey` column is currently just a dummy value of 0 or 1, and its sole purpose is to serve as a key for the truncate window. The `gdKey` column contains the value of the `gdKey` that the output adapter has successfully processed. The log window uses this to delete all the data that has a `gdKey` equal to or lesser than the provided value. In the `purge` column, a value of `true` indicates that the data in the log window needs to be deleted. The output adapter updates this column.

Assign this window to a log store to ensure data recovery from this window in the case of a failure.

Index

A

- adapter configuration 506
 - Open adapter 243
 - RAP adapter 306
- adapter controller parameters
 - FIX adapter 145
 - Flex adapter 180
 - HTTP adapter 193
 - NYSE adapter 230
 - TIBCO Rendezvous adapter 494
- adapter directory
 - FIX adapter 144
 - Flex adapter 179
 - HTTP adapter 191
 - NYSE adapter 228
 - TIBCO Rendezvous adapter 493
- adapter element 356, 383, 426, 458
- adapter logging 396, 469
- adapter operation
 - FIX adapter 164
 - Flex adapter 184
 - HTTP adapter 198
 - NYSE adapter 237
 - RAP adapter 318
 - RTView adapter 479
 - TIBCO Rendezvous adapter 501
- adapter property sets
 - creating 17
 - editing 17
- adapter schema
 - FIX adapter 145
 - Flex adapter 180
 - HTTP adapter 192
 - NYSEadapter 230
 - TIBCO Rendezvous adapter 494
- adapters 15
 - adapter shared utility library 505
 - adapter utilities 505
 - adding a new property set 17
 - AtomReader Input 18
 - ATTACH ADAPTER statement 3
 - attaching an adapter 4
 - basic steps for input adapters 3
 - basic steps for output adapters 4
 - configuring property sets 17
 - custom 3
 - custom external 517
 - custom internal 505
 - Database Input 20
 - Database Output 20, 22
 - external 127
 - File CSV Input 32
 - File CSV Output 35
 - File FIX Input 42
 - File FIX Output 44
 - File XML Input 37
 - File XML Output 41
 - FIX Input 147
 - guaranteed delivery 543
 - HTTP Output 196
 - information management functions 508
 - internal 18
 - introduction 1
 - JMS 46
 - JMS CSV Input 47
 - JMS CSV Output 50
 - JMS Custom Input 55
 - JMS Custom Output 59
 - JMS FIX Output 66
 - JMS Object Array Input 64, 70
 - JMS Object Array Output 74
 - JMS XML Input 79
 - JMS XML Output 82
 - KDB Input 204
 - KDB Output 209
 - life cycle functions 507
 - miscellaneous functions 508
 - NYSE Input 235
 - Open adapter directory 244
 - overview 1, 2
 - parameter datatypes 8
 - properties for schema discovery 539
 - publishing data 4
 - Random Tuples Generator Input 87
 - run states 509
 - schema discovery 539
 - SMTP Output 110
 - Socket (as Client) CSV Input 96
 - Socket (as Client) CSV Output 99
 - Socket (as Client) XML Input 101

Index

- Socket (as Client) XML Output 103
- Socket (as Server) CSV Input 107
- Socket (as Server) CSV Output 109
- Socket (as Server) XML Input 104
- Socket (as Server) XML Output 106
- Socket FIX Input 92
- Socket FIX Output 94
 - summary 15
 - supporting schema discovery 539
- Sybase IQ Output 115
- TIBCO Rendezvous 499
- WebSphere MQ Input 121
- WebSphere MQ Output 123
- administrative decisions 343, 416
 - Marketfeed Input adapter 332
 - OMM Input adapter 406
- all in one
 - sample configuration file 151
- APIs
 - supported languages 3
- applying a query
 - ESP Add-in for Microsoft Excel 134
- AsapSink
 - example 289
- AsapSource 246
 - example 291
- AsapSource properties 246
- ASE to ESP datatype mapping 327
- AtomReader Input adapter
 - internal adapter 18
 - properties 18
- ATTACH ADAPTER statement 3
- attaching an object to a cache
 - RTView adapter 482
- attaching an object to a stream
 - RTView adapter 483
- automatic publishing
 - ESP Add-in for Microsoft Excel 132
 - SybaseRTP function 132
- B**
- BeanShellPipe 253
 - example 292
- C**
- callback functions 506
- chain RICs 339
 - checking adapter status
 - FIX adapter 165
 - Flex adapter 185
 - HTTP adapter 199
 - NYSE adapter 238
 - TIBCO Rendezvous adapter 502
 - CLASSPATH environment variable 218
 - client socket connectors
 - FIX adapter 155
 - sample configuration file 156
 - cnxml file
 - custom internal adapters 506
 - column names
 - FIX adapter 141
 - configuration
 - creating a Sybase connection 476
 - data streams for NYSE adapter 232, 233
 - Flex Server 183
 - HTTP Server 195
 - Log File Input adapter 215
 - Marketfeed output adapter 342
 - OMM output adapter 415
 - Open adapter 243
 - property sets 17
 - RAP adapter 306, 313
 - Rendezvous Server 497
 - Replication Server adapter 320, 322
 - updating a Sybase connection 476
 - configuration file
 - FIX adapter 145
 - Flex adapter 180
 - HTTP adapter 192
 - NYSE adapter 230
 - TIBCO Rendezvous adapter 494
 - configuration files
 - RAP adapter 306
 - configuring a queuing system 46
 - configuring an input connection
 - from Reuters 328, 402
 - Reuters Marketfeed adapter 328
 - Reuters OMM adapter 402
 - configuring an output connection
 - Reuters Marketfeed adapter 330
 - Reuters OMM adapter 403
 - to Reuters 330, 403
 - connecting dashboard object to data streams
 - RTView adapter 481
 - Connection Wizard
 - ESP Add-in for Microsoft Excel 127

- constant element 384, 459
 - control flow
 - FIX adapter 136
 - Flex adapter 175
 - HTTP adapter 188
 - KDB adapter 202
 - NYSE adapter 219
 - TIBCO Rendezvous adapter 488
 - creating
 - OMM output adapter map file 418
 - creating a cache
 - RTView adapter 481
 - creating a dynamic watch list 340, 412
 - creating a function
 - RTView adapter 483
 - creating a Sybase connection 476
 - creating Marketfeed output adapter map file 344
 - creating shortcuts to Display Builder 480
 - creating shortcuts to Display Viewer 480
 - creating the input map file
 - Reuters Marketfeed Input adapter 336
 - Reuters OMM Input adapter 410
 - custom .Net external adapters
 - connecting to a subscriber 535
 - connecting to projects 534
 - connecting to the server 534
 - publishing data 534
 - subscribing using callback 535
 - custom adapters 505
 - overview 3
 - custom C/C++ external adapters
 - creating authentication credentials 531
 - getting a project 531
 - publishing and subscribing 531
 - sample code for handleData 533
 - subscribing using callback 531
 - custom external adapters 517
 - .Net adapters 534
 - auto-generated parameter files 525
 - configFilename parameter 526
 - datatypes 526
 - external adapter commands 521
 - external adapter configuration file 518
 - external adapter properties 521
 - parameter substitution 523
 - publish using callback 530
 - task overview 527
 - user-defined parameters 523
 - custom internal adapters 505
 - sample implementation 510
 - schema discovery 510
 - custom Java external adapters
 - connecting to projects 527
 - creating publishers 527
 - sample code for adding rows 528
 - subscribe using Direct Access mode 530
 - subscribing using callback 528
- ## D
- data decisions 342, 415
 - Marketfeed Input adapter 332
 - OMM Input adapter 405
 - data streams
 - FIX adapter 138
 - market data streams 224
 - NYSE adapter 224
 - order book data streams 225
 - TIBCO Rendezvous adapter 490
 - data streams configuration 233
 - data structures
 - Reuters Marketfeed adapter 333
 - Reuters OMM adapter 406
 - Database Input 20
 - Database Input adapter
 - properties 20
 - Database Output 20
 - Database Output adapter
 - properties 22
 - datafeed parameters
 - NYSE adapter 233
 - dataField element 357, 427
 - datatype formats for input adapters
 - date format 11
 - timestamp format 11
 - datatype formats for output adapters
 - date format 12
 - timestamp format 12
 - datatype mapping 26
 - ESP to ASE datatype mapping 327
 - ESP to FIX 143
 - ESP to KDB 202, 203
 - ESP to NYSE 228
 - ESP to Open adapter 242
 - ESP to RAP adapter 304
 - ESP to Replication Server datatype mapping 327
 - ESP to RTView 475

Index

- ESP to TIBCO Rendezvous 491
- File FIX Input adapter 43
- File FIX Output adapter 45
- IBM DB2 database 28
- KDB database 31
- KDB to ESP 203
- Microsoft SQL Server database 27
- Oracle database 29
- Replication Server adapter 327
- Socket FIX Input adapter 94
- Socket FIX Output adapter 96
- Sybase ASE database 26
- Sybase IQ Output adapter 120
- datatypes
 - adapter parameter datatypes 8
 - custom external adapters 526
 - supported datatypes in Event Stream Processor 5
- date format 11, 12
- datetime formats 266
- dateTimeField element 358, 429
- decisions
 - administrative 343, 416
 - data 342, 415
- delete
 - watchlist 240
- Display Builder 475
- Display Viewer 475
- dynamic watch lists
 - creating 340, 412
- E**
- enabling user access
 - Reuters Marketfeed adapter 328
 - Reuters OMM adapter 402
- encryption
 - Open adapter 287
- enum element 385
- environment variables 354, 424
 - CLASSPATH 218
 - FIX adapter 143
 - Flex adapter 179
 - HTTP adapter 191
 - NYSE adapter 228
 - Open adapter 243
 - TIBCO Rendezvous adapter 492
- ESP Add-in for Microsoft Excel
 - applying a query 134
 - automatic publishing 132
 - Connection Wizard 127
 - Publication Wizard 130
 - saving subscription queries 134
 - Subscription Wizard 129
 - SybaseRTP function 132
- ESP Add-In for Microsoft Excel
 - functionality 127
 - overview 127
- ESP Add-on for Microsoft Excel
 - known issues 135
 - limitations 135
- ESP datatype mapping
 - FIX adapter 143
 - KDB adapter 202
 - NYSE adapter 228
 - Open adapter 242
 - RAP adapter 304
 - RTView adapter 475
 - TIBCO Rendezvous adapter 491
- ESP to ASE datatype mapping 327
- ESP to Replication Server datatype mapping 327
- esp_ommsample 420
- esp_rmds 352
- esp_rmdsomm 422
- Event Stream Processor parameters
 - connecting to the Flex adapter 180
 - connecting to the HTTP adapter 193
 - connecting to the NYSE adapter 230
 - connecting to the TIBCO Rendezvous adapter 494
 - RTView adapter 477
- examples
 - configuring the RAP adapter 313
 - creating a function in RTView adapter 483
 - FIX adapter 139, 163, 166–168, 170, 173
 - Flex adapter 186
 - hosting inbound messages 163
 - HTTP adapter 200
 - NYSE adapter 240
 - Open adapter 289
 - receiving inbound messages 163
 - RTView adapter 485, 486
 - TIBCO Rendezvous adapter 503
 - using all in one 173
 - using AsapSink component 289
 - using AsapSource component 291
 - using BeanShellPipe component 292
 - using client socket connectors 168
 - using file connectors 167

- using JDBCLookupPipe component 293
 - using MultiFlatXmlStringReader component 295
 - using server socket connectors 170
 - using SpPersistentSubscribeSource component 296
 - using WSSink component 298
 - using WSSource component 299
 - using XPathMultiTypeXmlReader component 300
 - using XPathXmlStreamReader component 301
 - using XPathXMLStringWriter component 301
- external adapters 127
 - FIX Input 147
 - HTTP Output 196
 - KDB Input 204
 - KDB Output 209
 - NYSE Input 235
 - overview 2
 - TIBCO Rendezvous 499
- external data
 - input and output adapters 1
- F**
- FIDListField element 360
- field element 386, 460
- file connectors
 - FIX adapter 153
 - sample configuration file 154
- File CSV Input adapter
 - properties 32
- File CSV Output adapter
 - properties 35
- File FIX Input adapter
 - datatype mapping 43
 - properties 42
- File FIX Output adapter
 - datatype mapping 45
 - properties 44
- File XML Input adapter
 - properties 37
- File XML Output adapter
 - properties 41
- FIX adapter
 - adapter controller parameters 145
 - adapter directory 144
 - checking adapter status 165
 - client socket connectors 155
 - column names 141
 - configuration file 145
 - control flow 136
 - data streams 138
 - datatype mapping 143
 - Event Stream Processor Server properties 149
 - example 139, 163, 167, 168, 170, 173
 - file connectors 150, 153
 - FIX dictionary 149
 - header fields 141
 - hosting inbound messages 163
 - inbound connectors 151
 - log4j API 164
 - logging 164
 - login properties 162
 - message flow 142
 - operation 164
 - outbound connectors 151
 - overview 135
 - receiving inbound messages 163
 - record indexing 141
 - schema 145
 - sender login properties 162
 - server socket connectors 157
 - session connection properties 159–161
 - session login properties 162
 - session properties 163
 - sessions 141
 - socket connectors 150
 - start command 137
 - starting the adapter 164
 - status command 138
 - stop command 138
 - stopping the adapter 166
 - stream configuration 150
 - stream names 141
 - supported FIX protocol versions 136
 - trailer fields 141
- FIX adapter environment variables
 - JAVA_HOME 143
- FIX dictionary 149
- FIX Input adapter
 - properties 147
- FIX protocol versions 136
- FIX session properties 163
- Flex adapter
 - adapter controller parameters 180
 - adapter directory 179

Index

- checking adapter status 185
- client-server communication 177
- configuration file 180
- control flow 175
- Event Stream Processor parameters 180
- example 186
- log4j API 184
- logging 184
- message flow 176
- operation 184
- overview 175
- sample configuration file 183
- schema 180
- sending a subscription request 186
- start command 176
- starting the adapter 184
- status command 176
- stop command 176
- stopping the adapter 186
- Stream Handler 177
- subscribing to a stream 177
- Flex adapter environment variables
 - JAVA_HOME 179
- Flex Server settings 183
- Flex adapter
 - Flex Server settings 183
- formats for input adapters
 - date format 11
 - timestamp format 11
- formats for output adapters
 - date format 12
 - timestamp format 12
- G**
- generating self-signed RSA keys
 - Open adapter 288, 289
- getting stream information from project 344
- getting stream information from the project 417
- guaranteed delivery 543
 - log window 544
 - truncate window 545
- H**
- header fields
 - FIX adapter 141
- hiResTimestampField element 430
- HTTP adapter
 - adapter controller parameters 193
 - adapter directory 191
 - checking adapter status 199
 - configuration file 192
 - control flow 188
 - Event Stream Processor parameters 193
 - example 200
 - log4j API 197
 - logging 197
 - message flow 190
 - operation 198
 - overview 188
 - receiving data 200
 - sample configuration file 195
 - schema 192
 - sending data 200
 - start command 189
 - starting the adapter 198
 - status command 189
 - stop command 189
 - stopping the adapter 199
 - viewing data 200
- HTTP adapter environment variables
 - JAVA_HOME 191
- HTTP Output adapter
 - properties 196
- HTTP Server settings 195
- I**
- IBM DB2 database
 - datatype mapping 28
- imageField element 431
- inbound connectors
 - FIX adapter 151
- individual RICs 339, 411
- input adapter
 - map file 355, 426
- input adapter map file
 - Reuters Marketfeed Input adapter 333
 - Reuters OMM Input adapter 406
- input adapters 499
 - AtomReader Input 18
 - Database Input 20
 - File CSV Input 32
 - File XML Input 37
 - FIX Input 147
 - JMS CSV Input 47
 - JMS Custom Input 55
 - JMS Object Array Input 64, 70
 - JMS XML Input 79

- KDB Input 204
- KDB Output 209
- NYSE Input 235
- overview 1
- Random Tuples Generator Input 87
- Socket (as Client) CSV Input 96
- Socket (as Client) XML Input 101
- Socket (as Server) CSV Input 107
- Socket (as Server) XML Input 104
- Socket FIX Input 92
- WebSphere MQ Input 121
- insert
 - watchlist 239
- installation
 - RTView adapter 476
- internal adapter
 - WebSphere MQ adapter 120
- internal adapters 18
 - AtomReader Input 18
 - Database Input 20
 - Database Output 22
 - File CSV Input 32
 - File CSV Output 35
 - File FIX Input 42
 - File FIX Output 44
 - File XML Input 37
 - File XML Output 41
 - JMS CSV Input 47
 - JMS CSV Output 50
 - JMS Custom Input 55
 - JMS Custom Output 59
 - JMS FIX Output 66
 - JMS Object Array Input 64, 70
 - JMS Object Array Output 74
 - JMS XML Input 79
 - JMS XML Output 82
 - overview 2
 - Random Tuples Generator Input 87
 - SMTP Output 110
 - Socket (as Client) CSV Input 96
 - Socket (as Client) CSV Output 99
 - Socket (as Client) XML Input 101
 - Socket (as Client) XML Output 103
 - Socket (as Server) CSV Input 107
 - Socket (as Server) CSV Output 109
 - Socket (as Server) XML Input 104
 - Socket (as Server) XML Output 106
 - Socket FIX Input 92
 - Socket FIX Output 94
 - Sybase IQ Output 115
 - WebSphere MQ Input 121
 - WebSphere MQ Output 123
 - item element 361, 432
 - itemList element 363, 434
 - itemLists element 364, 436
 - itemName element 366, 437
 - itemStale element 367, 439
- J**
 - JAVA_HOME environment variable
 - FIX adapter 143
 - Flex adapter 179
 - HTTP adapter 191
 - NYSE adapter 228
 - Open adapter 243
 - TIBCO Rendezvous adapter 492
 - JDBCLookupPipe 253
 - example 293
 - JMS adapter 46
 - configuring a queuing system 46
 - JMS CSV Input adapter
 - properties 47
 - JMS CSV Output adapter
 - properties 50
 - JMS Custom Input adapter
 - properties 55
 - JMS Custom Output adapter
 - properties 59
 - JMS FIX Output adapter
 - properties 66
 - JMS Object Array Input adapter
 - properties 64, 70
 - JMS Object Array Output adapter
 - properties 74
 - JMS XML Input adapter
 - properties 79
 - JMS XML Output adapter
 - properties 82
- K**
 - KDB adapter
 - control flow 202
 - datatype mapping 202
 - ESP to KDB datatype mapping 203
 - KDB to ESP datatype mapping 203
 - overview 202

Index

- start command 202
- stop command 202
- KDB database
 - datatype mapping 31
- KDB Input adapter
 - properties 204
- KDB Output adapter
 - properties 209
- known limitations
 - RTView adapter 487
- L**
- life cycle functions 507
- Log File Input adapter
 - CLASSPATH environment variable 218
 - configuration 215
 - overview 214
 - properties 216
 - starting the adapter 218
- log messages 399, 473
- log4j API
 - Flex adapter 184
 - HTTP adapter 197
 - NYSE adapter 236
 - TIBCO Rendezvous adapter 500
- logging
 - adapter 396, 469
 - FIX adapter 164
 - Flex adapter 184
 - HTTP adapter 197
 - log4j API 164
 - NYSE adapter 236
 - Reuters 399, 472
 - TIBCO Rendezvous adapter 500
- logging facilities 395, 469
- M**
- map file
 - creating a subordinate map file 348, 420
 - creating the input map file 336, 410
 - input adapter 355, 426
 - modifying the main map file 348, 420
 - output adapter 382
 - Reuters Marketfeed Input adapter 333
 - Reuters OMM Input adapter 406
- market data field mapping
 - Reuters Marketfeed adapter 333
 - Reuters OMM adapter 407
- market data streams 224
- market data watchlists 222
- marketByOrderKeyField element 441
- marketByPriceKeyField element 442
- Marketfeed Input adapter
 - administrative decisions 332
 - data decisions 332
- Marketfeed output adapter
 - configuration 342
 - running 347
 - testing 347
- Marketfeed output adapter map file
 - creating 344
- message flow
 - FIX adapter 142
 - Flex adapter 176
 - HTTP adapter 190
 - NYSE adapter 227
 - TIBCO Rendezvous adapter 490
- Microsoft SQL Server database
 - datatype mapping 27
- model files 506
- MultiFlatXmlStringReader 257
 - example 295
- N**
- name element 387, 461
- nullField element 368, 444
- NYSE adapter
 - adapter controller parameters 230
 - adapter directory 228
 - checking adapter status 238
 - configuration file 230
 - control flow 219
 - data stream configuration 233
 - data streams 224
 - datafeed parameters 233
 - datatype mapping 228
 - Event Stream Processor parameters 230
 - example 240
 - log4j API 236
 - logging 236
 - market data streams 224
 - market data watchlists 222
 - message flow 227
 - modifying watchlists 239
 - operation 237
 - order book data streams 225

- order book watchlists 223
 - overview 219
 - publishing data 240
 - sample configuration file 234
 - schema 230
 - stale data stream records 226
 - start command 220
 - starting the adapter 237
 - status command 221
 - stop command 220
 - stopping the adapter 239
 - subscribing to data 240
 - watchlist delete 240
 - watchlist insert 239
 - watchlist stream configuration 232
 - watchlists 221, 239
 - NYSE adapter environment variables
 - JAVA_HOME 228
 - NYSE Input adapter
 - properties 235
- O**
- OMM adapter output map file
 - creating 418
 - OMM Input adapter
 - administrative decisions 406
 - data decisions 405
 - OMM output adapter
 - configuration 415
 - performance tuning 419
 - running 418
 - testing 418
 - Open adapter
 - Africa time zones 270
 - AsapSink example 289
 - AsapSink properties 250
 - AsapSource 246
 - AsapSource example 291
 - AsapSource properties 246
 - Asia time zones 272
 - Australasia time zones 275
 - BeanShellPipe example 292
 - BeanShellPipe properties 253
 - configuration 243
 - datatype mapping 242
 - directory 244
 - encryption 287
 - EspDelimitedStringReader properties 263
 - Europe time zones 277
 - examples 289
 - generating self-signed RSA keys 288, 289
 - HTTPRemoteControl 286
 - JDBCLookupPipe example 293
 - JDBCLookupPipe properties 255
 - MailRemoteLogger 287
 - MultiFlatXmlStringReader example 295
 - MultiFlatXmlStringReader properties 257
 - North America time zones 279
 - overview 241
 - PasswordEncryptor 287
 - reader components 257
 - remote control attributes 285
 - remote control interface 285
 - remote control methods 285
 - RemoteControl interface 284
 - RemoteLogger interface 284
 - sample key stores 287
 - sink components 250
 - source components 246
 - South America time zones 282
 - specifying datetime formats 266
 - SpPersistentSubscribeSource 246
 - SpPersistentSubscribeSource example 296
 - SpPersistentSubscribeSource properties 248
 - starting the adapter 284
 - third-party JAR files 267
 - time zones 269
 - WSSink example 298
 - WSSink properties 252
 - WSSource example 299
 - XPathMultiTypeXmlReader example 300
 - XPathMultiTypeXmlReader properties 262
 - XPathXmlStreamReader example 301
 - XPathXmlStreamReader properties 259
 - XPathXMLStringWriter example 301
 - XPathXmlStringWriter properties 264
 - XPathXmlWriter 264
 - Open Adapter
 - pipe components 253
 - Open adapter components 246
 - Open adapter environment variables
 - JAVA_HOME 243
 - Open Adapter pipe components
 - BeanShellPipe 253
 - JDBCLookupPipe 253
 - Open adapter reader components
 - MultiFlatXmlStringReader 257
 - XPathMultiTypeXmlReader 257

Index

- XPathXmlStreamReader 257
- Open adapter sink components
 - AsapSink 250
 - WSSink 250
- Open adapter source components
 - AsapSource 246
 - SpPersistentSubscribeSource 246
- Open adapter writer component
 - XPathXmlWriter 264
- operating systems 327
- operation
 - FIX adapter 164
 - Flex adapter 184
 - HTTP adapter 198
 - NYSE adapter 237
 - RAP adapter 318
 - RTView adapter 479
 - TIBCO Rendezvous adapter 501
- Oracle database
 - datatype mapping 29
- order book data streams 225
- order book watchlists 223
- outbound connectors
 - FIX adapter 151
- output adapter
 - map file 382
- output adapters
 - Database Output 22
 - File CSV Output 35
 - File FIX Input 42
 - File FIX Output 44
 - File XML Output 41
 - HTTP Output 196
 - JMS CSV Output 50
 - JMS Custom Output 59
 - JMS FIX Output 66
 - JMS Object Array Output 74
 - JMS XML Output 82
 - overview 1
 - SMTP Output 110
 - Socket (as Client) CSV Output 99
 - Socket (as Client) XML Output 103
 - Socket (as Server) CSV Output 109
 - Socket (as Server) XML Output 106
 - Socket FIX Output 94
 - Sybase IQ Output 115
 - TIBCO Rendezvous 499
 - WebSphere MQ Output 123

- overview 327
 - FIX adapter 135
 - Flex adapter 175
 - HTTP adapter 188
 - KDB adapter 202
 - Log File Input adapter 214
 - NYSE adapter 219
 - Open adapter 241
 - RAP adapter 303
 - Replication Server Adapter 320
 - RTView adapter 475
 - TIBCO Rendezvous adapter 488

P

- PasswordEncryptor
 - Open adapter 287
- performance tips
 - Replication Server adapter 327
- performance tuning
 - OMM output adapter 419
 - Reuters Marketfeed Input adapter 350
 - Reuters OMM Input adapter 413
- persistent subscribe pattern 543
- pipe components
 - BeanShellPipe 253
 - JDBCLookupPipe 253
- properties
 - AsapSink 250
 - AsapSource component 246
 - AtomReader Input adapter 18
 - BeanShellPipe 253
 - Database Input adapter 20
 - Database Output adapter 22
 - EspDelimitedStringReader 263
 - File CSV Input adapter 32
 - File CSV Output adapter 35
 - File FIX Input adapter 42
 - File FIX Output adapter 44
 - File XML Input adapter 37
 - File XML Output adapter 41
 - FIX adapter 159–162
 - FIX Input adapter 147
 - FIX sessions 163
 - HTTP Output adapter 196
 - JDBCLookupPipe 255
 - JMS CSV Input adapter 47
 - JMS CSV Output adapter 50
 - JMS Custom Input adapter 55
 - JMS Custom Output adapter 59

- JMS FIX Output adapter 66
 - JMS Object Array Input adapter 64, 70
 - JMS Object Array Output adapter 74
 - JMS XML Input adapter 79
 - JMS XML Output adapter 82
 - KDB Input adapter 204
 - KDB Output adapter 209
 - Log File Input adapter 216
 - MultiFlatXmlStreamReader 257
 - NYSE Input adapter 235
 - Random Tuples Generator Input adapter 87
 - schema discovery 539
 - SMTP Output adapter 110
 - Socket (as Client) CSV Input adapter 96
 - Socket (as Client) CSV Output adapter 99
 - Socket (as Client) XML Input adapter 101
 - Socket (as Client) XML Output adapter 103
 - Socket (as Server) CSV Input adapter 107
 - Socket (as Server) CSV Output adapter 109
 - Socket (as Server) XML Input adapter 104
 - Socket (as Server) XML Output adapter 106
 - Socket FIX Input adapter 92
 - Socket FIX Output adapter 94
 - SpPersistentSubscribeSource 248
 - Sybase IQ Output adapter 115
 - TIBCO Rendezvous adapter 499
 - WebSphere MQ Input adapter 121
 - WebSphere MQ Output adapter 123
 - WSSink 252
 - XPathMultiTypeXmlReader 262
 - XPathXmlStreamReader 259
 - XPathXmlStringWriter 264
 - property sets
 - creating 17
 - editing 17
 - publication element 370, 446
 - Publication Wizard
 - ESP Add-in for Microsoft Excel 130
 - publisher file
 - RAP adapter 308
- Q**
- queue configuration
 - WebSphere adapter 126
- R**
- Random Tuples Generator Input adapter
 - properties 87
 - RAP adapter
 - configuration 306
 - configuration file 306
 - configuring the RAP adapter 313
 - datatype mapping 304
 - example configuration 313
 - operation 318
 - overview 303
 - publisher file 308
 - RDS template file 310
 - start command 303
 - starting the adapter 319
 - stop command 304
 - stopping the adapter 319
 - RDS template file
 - RAP adapter 310
 - reader components
 - MultiFlatXmlStreamReader 257
 - XPathMultiTypeXmlReader 257
 - XPathXmlStreamReader 257
 - record indexing
 - FIX adapter 141
 - recordType element 371
 - recordTypeMap element 372
 - remote control methods
 - Open adapter 285
 - Rendezvous Server settings 497
 - Replication Server adapter
 - configuration 320, 322
 - datatype mapping 327
 - performance tips 327
 - Replication Server Adapter
 - defining a persistent table 326
 - overview 320
 - rs_lastcommit table 326
 - Replication Server to ESP datatype mapping 327
 - respTypeNumField element 448
 - Reuters information 343, 416
 - Reuters Instrument Codes 334, 407
 - Reuters logging 399, 472
 - Reuters Marketfeed adapter
 - data structures 333
 - enabling user access 328
 - input connection 328
 - market data field mapping 333
 - output connection 330
 - Reuters Instrument Codes 334
 - testing the adapter 338

Index

- Reuters Marketfeed Input adapter
 - creating the input map file 336
 - map file 333
 - performance tuning 350
 - running the adapter 338
 - Reuters OMM adapter
 - data structures 406
 - enabling user access 402
 - input connection 402
 - market data field mapping 407
 - output connection 403
 - Reuters Instrument Codes 407
 - testing the adapter 411
 - Reuters OMM Input adapter
 - creating the input map file 410
 - map file 406
 - performance tuning 413
 - running the adapter 410
 - rfa element 374, 388, 449, 463
 - RSA keys
 - generating RSA keys for Open adapter 288, 289
 - RTView adapter
 - attaching an object to a cache 482
 - attaching an object to a stream 483
 - components 475
 - configuration 476
 - connecting dashboard object to data streams 481
 - creating a cache 481
 - creating a Sybase connection 476
 - creating shortcuts to Display Builder 480
 - creating shortcuts to Display Viewer 480
 - datatype mapping 475
 - Event Stream Processor parameters 477
 - installing the adapter 476
 - known limitations 487
 - operation 479
 - overview 475
 - publishing data 484
 - running the publisher example 485
 - running the subscriber example 486
 - starting the adapter 479, 480
 - updating a Sybase connection 476
 - RTView Display Viewer
 - running the publisher example 485
 - running the adapter
 - Reuters Marketfeed Input adapter 338
 - Reuters OMM Input adapter 410
 - running the Marketfeed output adapter 347
 - running the OMM output adapter 418
 - running the publisher example for RTView adapter 485
- ## S
- sample configuration file
 - all in one 151
 - client socket connectors 156
 - file connectors 154
 - sample configuration files
 - Flex adapter 183
 - HTTP adapter 195
 - NYSE adapter 234
 - server socket connectors 158
 - TIBCO Rendezvous adapter 498
 - saving subscription queries
 - ESP Add-in for Microsoft Excel 134
 - schema
 - adapters 539
 - discovery 539
 - schema discovery
 - adapter properties 539
 - adapters that support it 539
 - overview 539
 - SDKs
 - supported languages 3
 - sender login properties
 - FIX adapter 162
 - sequenceNumber element 375, 451
 - server socket connectors
 - FIX adapter 157
 - sample configuration files 158
 - service element 390
 - serviceName element 377, 452
 - session connection properties
 - FIX adapter 159–161
 - session login properties
 - FIX adapter 162
 - sink components
 - AsapSink 250
 - WSSink 250
 - SMTP Output adapter
 - properties 110
 - Socket (as Client) CSV Input adapter
 - properties 96
 - Socket (as Client) CSV Output adapter
 - properties 99

- Socket (as Client) XML Input adapter
 - properties 101
 - Socket (as Client) XML Output adapter
 - properties 103
 - Socket (as Server) CSV Input adapter
 - properties 107
 - Socket (as Server) CSV Output adapter
 - properties 109
 - Socket (as Server) XML Input adapter
 - properties 104
 - Socket (as Server) XML Output adapter
 - properties 106
 - Socket FIX Input adapter
 - datatype mapping 94
 - properties 92
 - Socket FIX Output adapter
 - datatype mapping 96
 - properties 94
 - source components
 - AsapSource 246
 - SpPersistentSubscribeSource 246
 - specifying datetime formats
 - Open adapter 266
 - SpPersistentSubscribeSource 246
 - example 296
 - SpPersistentSubscribeSource properties 248
 - stale data stream records
 - NYSE adapter 226
 - stale element 391, 464
 - start command
 - FIX adapter 137
 - Flex adapter 176
 - HTTP adapter 189
 - KDB adapter 202
 - NYSE adapter 220
 - RAP adapter 303
 - TIBCO Rendezvous adapter 489
 - starting an adapter
 - FIX adapter 164
 - Flex adapter 184
 - HTTP adapter 198
 - Log File Input adapter 218
 - NYSE adapter 237
 - Open adapter 284
 - RAP adapter 319
 - RTView adapter 479, 480
 - TIBCO Rendezvous adapter 501
 - statements
 - ATTACH ADAPTER statement 3
 - status command
 - FIX adapter 138
 - Flex adapter 176
 - HTTP adapter 189
 - NYSE adapter 221
 - TIBCO Rendezvous adapter 490
 - stop command
 - FIX adapter 138
 - Flex adapter 176
 - HTTP adapter 189
 - KDB adapter 202
 - NYSE adapter 220
 - RAP adapter 304
 - TIBCO Rendezvous adapter 489
 - stopping the adapter
 - FIX adapter 166
 - Flex adapter 186
 - HTTP adapter 199
 - NYSE adapter 239
 - RAP adapter 319
 - TIBCO Rendezvous adapter 502
 - stream configuration
 - FIX adapter 150
 - stream element 392, 466
 - Stream Handler
 - Flex adapter 177
 - stream information
 - getting from project 344
 - getting from the project 417
 - stream names
 - FIX adapter 141
 - streamMap element 378, 453
 - streamMaps element 380, 455
 - streams
 - schema discovery 539
 - subscription element 393, 467
 - Subscription Wizard
 - ESP Add-in for Microsoft Excel 129
 - subscriptions element 395, 468
 - supported operating systems 327
 - Sybase ASE database
 - datatype mapping 26
 - Sybase IQ Output adapter
 - datatype mapping 120
 - properties 115
- ## T
- testing the adapter
 - Reuters Marketfeed adapter 338

Index

- Reuters OMM adapter 411
- testing the Marketfeed output adapter 347
- testing the OMM output adapter 418
- third-party JAR files
 - Open Adapter 267
- Tibco Rendezvous adapter
 - HTTP Server settings 195
- TIBCO Rendezvous adapter
 - adapter controller parameters 494
 - adapter directory 493
 - checking adapter status 502
 - configuration file 494
 - control flow 488
 - data streams 490
 - datatype mapping 491
 - Event Stream Processor parameters 494
 - example 503
 - input stream parameters 496
 - log4j API 500
 - logging 500
 - message flow 490
 - operation 501
 - output stream properties 497
 - overview 488
 - properties 499
 - publishing data 503
 - Rendezvous Server settings 497
 - sample configuration file 498
 - schema 494
 - start command 489
 - starting the adapter 501
 - status command 490
 - stop command 489
 - stopping the adapter 502
 - uploading records 503
- TIBCO Rendezvous adapter environment variables
 - JAVA_HOME 492
- time zones for Open adapter 269
 - Africa 270
 - Asia 272
 - Australasia 275
 - Europe 277
 - North America 279
 - South America 282
- timestamp format 11, 12
- trailer fields
 - FIX adapter 141

U

- updateNumber element 381, 457
- updating a Sybase connection 476

V

- variables
 - environment 354, 424

W

- watchlist delete 240
- watchlist insert 239
- watchlist stream configuration 232
- watchlists 239
 - market data watchlists 221, 222
 - order book watchlists 221, 223
- WebSphere adapter
 - queue configuration 126
- WebSphere MQ adapter
 - overview 120
- WebSphere MQ Input adapter
 - properties 121
- WebSphere MQ Output adapter
 - properties 123
- windows
 - schema discovery 539
- writer component
 - XPathXmlWriter 264
- WSSink
 - example 298
- WSSource
 - example 299

X

- XPathMultiTypeXmlReader 257
 - example 300
- XPathXmlStreamReader 257
 - example 301
- XPathXMLStringWriter
 - example 301
- XPathXmlWriter 264