
Micro Focus Security ArcSight Connectors

FlexConnector

Developer's Guide

Document Release Date: July 24, 2019

Software Release Date: July 24, 2019



Legal Notices

Warranty

The only warranties for products and services of Micro Focus and its affiliates and licensors (“Micro Focus”) are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Micro Focus shall not be liable for technical or editorial errors or omissions contained herein. The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Except as specifically indicated otherwise, a valid license from Micro Focus is required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the US Government under vendor's standard commercial license.

Copyright Notice

©copyright 2018 Micro Focus

Trademark Notices

Adobe™ is a trademark of Adobe Systems Incorporated.

Microsoft® and Windows® are US registered trademarks of Microsoft Corporation.

UNIX® is a registered trademark of The Open Group.

Support

Contact Information

Phone	A list of phone numbers is available on the Technical Support Page: https://softwaresupport.softwaregrp.com/support-contact-information
Support Web Site	https://softwaresupport.softwaregrp.com/
ArcSight Product Documentation	https://communitysoftwaregrp.com/t5/ArcSight-Product-Documentation/ct-p/productdocs

Revision History

Date	Description
04/16/2018	<ul style="list-style-type: none"> Added configuration properties for JSON Multiple Folder Follower FlexConnector.
10/17/2017	<ul style="list-style-type: none"> Updated "Set Global Parameters" section to include encryption parameters. Updated information for downloading SQL Server JDBC drivers. In Appendix E: ArcSight Built-in Event Field Mappings, the ArcSight Mappings fields have been changed to camel case. Only non-blocking I/O is available for syslog connectors; therefore, the tcpmaxidletime, tcpsetsocketlinger, and tcppeerclosedchecktimeout parameters are no longer relevant and have been removed from the Advanced Parameters appendix. As flexString fields are for the use of customers, examples have been updated to show deviceCustomString or deviceCustomNumber fields rather than flexString fields.
05/15/2017	<ul style="list-style-type: none"> Added a notice about ODBC connections not being supported after release 7.2.1 to the "ArcSight FlexConnector ID-Based Database", "ArcSight FlexConnector Multiple Database", and "ArcSight FlexConnector Scanner Database" sections.
02/15/2017	<ul style="list-style-type: none"> Added JSON to the list of available extra processors. See "Extra Processors". Clarified the configuration file names and locations for vulnerabilities, open ports, and URIs for scanner FlexConnectors for normal text reports. See "Getting Vulnerabilities for Scanned Hosts", "Getting Open Ports on Scanned Hosts", and "Getting OS and Applications (URIs) on Scanned Hosts".
11/30/2016	<ul style="list-style-type: none"> Updated installation procedure for setting preferred IP address mode. Updated FlexConnector information for IPv6-aware parsers.
08/30/2016	<ul style="list-style-type: none"> Reorganized and expanded content for increased usability. Updated the "Configure the JDBC Driver and Windows Authentication" section. In "Advanced Parameters", updated information regarding preservestate parameters.
06/30/2016	<ul style="list-style-type: none"> Added parameters to "Parameters Common to all SmartConnectors".

Date	Description
05/16/2016	<ul style="list-style-type: none"> • Updates and clarifications in the “Log Rotation Types” section. Added a section on the unparsed events detection feature: “Unparsed Events Detection”. • In “Advanced Parameters”, clarified the descriptions of several advanced parameters.
03/31/2016	<ul style="list-style-type: none"> • Added advanced parameters to customize connector behavior as Chapter 5. • In “Advanced Parameters”, noted that for Syslog connectors, the persistenceinterval parameter must be a positive integer to enable persistence. • In “Advanced Parameters”, noted that the rawlogfolder and usefilequeue parameters cannot be applied to Syslog Pipe/File Connector. • Added configuration properties for JSON Folder Follower FlexConnector/JSON Multiple Folder Follower FlexConnector. • Noted that only one question mark is supported for time-based database FlexConnector queries. • Removed agents[x].maxfilesize parameter.
02/15/2016	<ul style="list-style-type: none"> • End of life for FlexConnector SNMP (install the SmartConnector for SNMP Unified). • Added the new feature to detect and log unparsed events. • Updated the time format for __parseMutableTimeStamp function. • Updated the wildcard parameter default value to use *.

Contents

Chapter 1: Overview	14
FlexConnector Development	14
IPv6-Aware Parsers	15
Event Fields	15
Operations	15
Developer Considerations	16
Folder Structure	16
Key Files	17
FlexConnector Management	17
ArcSight Connector Appliance	17
ArcSight Management Center	18
Chapter 2: Choose a FlexConnector Type	20
FlexConnector Types	20
Event Data Format Examples	22
Log File FlexConnector	22
ID-Based Database FlexConnector	22
JSON Folder Follower FlexConnector/JSON Multiple Folder Follower FlexConnector	23
Multiple Database FlexConnector	24
Regex FlexConnectors (Variable-Format File FlexConnectors)	24
Scanner FlexConnector	25
SNMP FlexConnector	25
Syslog FlexConnector	27
Time-Based Database FlexConnector	28
XML File FlexConnector	28
Chapter 3: Install and Configure the FlexConnector	31
FlexConnector Installation	31
Install Core Software	31
Set Global Parameters (Optional)	32
Select Connector and Add Parameter Information	33
ArcSight FlexConnector File	33
ArcSight FlexConnector ID-Based Database	34
ArcSight FlexConnector JSON Multiple Folder Follower	37

ArcSight FlexConnector Multiple Database	39
ArcSight FlexConnector Multiple Folder File	43
ArcSight FlexConnector Regex File	44
ArcSight FlexConnector Regex Folder File	45
ArcSight FlexConnector REST	46
ArcSight FlexConnector Scanner Database	47
ArcSight FlexConnector Scanner Text Reports	50
ArcSight FlexConnector Scanner XML Reports	51
ArcSight FlexConnector XML File	53
ArcSight FlexConnector Simple Network Management Protocol (SNMP Unified)	54
ArcSight FlexConnector Syslog	54
Select a Destination	55
Complete Installation and Configuration	56
Additional Configuration for Database Connectors	56
Install SQL Server JDBC Driver	57
Install MySQL Driver	58
Add a JDBC Driver to the Connector Appliance/ArcSight Management Center	59
Configure the JDBC Driver and Windows Authentication	60
Oracle 8i Support	61
Troubleshooting Duplicate Events	61
Example 1: ID-based Database Connectors Only	61
Example 2: ID-based and Time-based Connectors	62
Example 3: Complex Main Query with a Join	62
Chapter 4: Create a Configuration File	65
Parser File Locations and Names	65
Example Parser File	66
Parser File Structure	67
Token Declarations	68
Token Types	69
Event Mapping	69
HttpRequest Event Field	69
Operations Table	70
Severity Mapping	71
Examples	72
Extra Processors	72
Key-Value Parsers	74
FlexConnector Creation Wizard for Delimited Log Files	75

Regex Tool for Regex FlexConnectors	78
Start the FlexConnector	81
Chapter 5: Configuration File Examples	82
Configuration Properties for a Log File FlexConnector	82
Configuration Properties for all Regex FlexConnectors	83
Configuration Properties for a Time-based Database FlexConnector	84
Version	84
Query	85
Timestamp	86
UniqueID	86
Configuration Properties for an ID-based Database FlexConnector	86
Version	87
MaxID	87
Query	87
ID	88
UniqueID	88
Query Limit	88
Configuration Properties for an SNMP Connector	88
Configuration Properties for an XML FlexConnector	90
Namespace	91
Hop Nodes	91
Trigger Nodes	91
Token Mappings	92
Examples of Token Mappings	92
Extra Events	93
Configuration Properties for a JSON Folder Follower FlexConnector/ JSON Multiple Folder Follower FlexConnector	93
Trigger Node	95
Token Location and Mappings	95
JSON Parsers for Complex Event Schemas	95
Working with Hierarchical Schemas	95
Representing a JSON Array with a Key Element	97
Representing a Token Value in URI Format	98
Sample JSON Array	99
Configuration Properties for Scanner FlexConnectors	99
Scanner FlexConnectors for Normal Text or XML Scan Reports	99

How Scanner FlexConnectors Parse Scan Reports	100
Parser Files for Normal Text Reports	100
Getting a List of Hosts	101
Ignore or Include Line	101
Regular Expression and Token Mappings	102
Use IP	102
Invalid Vulnerabilities	103
Extra Events	103
Getting Vulnerabilities for Scanned Hosts	104
Token Mappings	105
Event Mappings	105
Severity Mappings	106
Ignore or Include Line	106
Getting Open Ports on Scanned Hosts	107
Token Mappings	108
Event Mappings	108
Ignore or Include Line	108
Getting OS and Applications (URLs) on Scanned Hosts	109
Token Mappings	109
Event Mappings	110
Ignore or Include Line	110
Configuration Files for XML Reports	110
Getting a List of Hosts	110
Token Mappings	111
Use IP	111
Invalid Vulnerabilities	111
Extra Events	112
Getting Vulnerabilities for Scanned Hosts	112
Token Mappings	114
Event Mappings	114
Severity Mappings	115
Getting Open Ports on Scanned Hosts	115
Token Mappings	117
Event Mappings	117
Getting OS and Applications (URLs) on Scanned Hosts	117
Token Mappings	118
Event Mappings	118
Scanner FlexConnectors for Database Scan Reports	119
Getting the Version of the Database	119
Version	119

Getting the List of Scan Jobs	120
Scan Job	120
Use IP	120
Invalid Vulnerabilities	120
Extra Queries	121
Vulnerability Query	121
Open Ports Query	124
Getting OS and Applications (URLs) on Scanned Hosts	125
Getting Scanned Hosts (Host Query)	125
Chapter 6: Advanced Features	127
Regular Expressions	127
Multi-line Parsing	129
Sub-Messages	132
Default Sub-message	138
Extra Mappings	139
Conditional Mappings	141
Using Conditional Mapping in Sub-messages	143
Additional Data Mapping	144
Using the Get Additional Data Names Command	144
Using the Map Additional Data Name... Command	145
Using the Unmap Additional Data Name... Command	146
Using the Get Status Command	147
Log Rotation Types	147
Name Following Log Rotation	147
Daily Rotation	148
Index Rotation	148
Parameters for Daily and Index Rotation	148
Using rotationschemeparams for Daily Log File Rotation	148
Using rotationschemeparams for Index Log File Rotation	150
Using wildcard for Daily and Index Log File Rotation (File Folder Follower Only)	150
Using wildcard for Date Rotation	150
Using wildcard for Index Rotation	151
Log Internal Events for File-Reading FlexConnectors	151
Unparsed Events Detection	152
Supported Parser Types	152
Unparsed Events Detection Criteria	153
Comment Expressions	155

Parsing Expressions	156
Token Expressions	156
Mapping Expressions	157
Extra-Processor Expressions	157
Criteria for Unparsed Events	158
Unparsed Events Output File	160
Chapter 7: Map Files	162
What Are Map Files?	162
Map File Examples	162
Multiple "Getters" and "Setters"	163
Using the "No Getter" Trick	164
Map File Details	164
Controlling Map File Operation	164
Basic Map Files	165
AgentInfoAdder1 Map Files	166
Categorizer Map Files	166
Extra Processor Map Files	167
Using Ranges in Map Files	167
Using Regular Expressions in Map Files	168
Using Parser-Like Expressions in Map Files	169
More About Parser-Like Expressions Syntax	170
Operations Containing Commas	170
Backslashes in Expressions Versus in Parsers	170
Real World Examples	171
Adding Country Names to Events	171
Getting Domain Name from Hostname	171
Appendix A: ArcSight Operations	173
Appendix B: ArcSight Built-in Tokens	187
Appendix C: ArcSight Built-in Token Types	188
Appendix D: Date and Time Format Symbols	189
Appendix E: ArcSight Built-in Event Field Mappings	190

Appendix F: Configuring a Connector for ArcSight ESM Domain Field Sets	196
Appendix G: Advanced Parameters	199
Parameters Common to all SmartConnectors	200
CEF Syslog Parameters	202
File Connector Parameters	204
File Folder Follower Parameters	206
Syslog Parameters	210
Syslog Daemon Parameters	211
Event Parsing (Sub-agents) Parameters	211
Event Reception Parameters	212
Raw Log Parameters	214
Event Queue Parameters	215
Event Processing Parameters	216
Syslog Pipe Parameters	217
Syslog File Parameters	217
Syslog NG Daemon Parameters	219
Raw Syslog Daemon Parameters	219
ArcSight CEF Encrypted Syslog (UDP) Parameters	220
TippingPoint SMS Syslog Extended Parameters	220
Appendix H: FlexConnectors and Categorization	221
Categorization	221
HTTP Status Code Categorization Example	221
Firewall Example	224
Appendix I: Developing a Syslog FlexConnector	225
Appendix J: Developing an XML FlexConnector	227
XML FlexConnector Development	227
XML Tools	227
XML Concepts for FlexConnector Development	228
General XML Concepts	228
XML FlexConnector Concepts	229
Namespace	229
Hop Nodes	229

Trigger Nodes	230
Token Mappings	230
Extra Events	230
Examples of Token Mappings	231
Prepare to Write the Parser - Identify Namespace, Nodes, and Tokens	231
Find the Trigger Node - the Most Important Step	232
Decide if You Need a Namespace	232
Identify Hop Nodes	233
Identify Tokens	233
Create the XML FlexConnector Parser	234
Parser Development - First Several Lines	234
Parser Development Continued - Tokens	235
Parser Development Continued - Mappings	235
Categorization	236
Copy the Parser Into the Folder	236
Install the FlexConnector	237
Appendix K: Frequently Asked Questions	238
Send Documentation Feedback	242

Document Revision History

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.

To check for recent updates or to verify that you are using the most recent edition of a document, go to [ArcSight Product Documentation Community on the Micro Focus Security Community](#).

Document Changes

Date	Product Version	Description

About this PDF Version of Online Help

This document is a PDF version of the online help. This PDF file is provided so you can easily print multiple topics from the help information or read the online help in PDF format. Because this content was originally created to be viewed as online help in a web browser, some topics may not be formatted properly. Some interactive topics may not be present in this PDF version. Those topics can be successfully printed from within the online help.

Chapter 1: Overview

Security ArcSight provides a range of device-specific SmartConnectors with which to gather security event information. The connectors send normalized security events to the specified destination for storage and further processing. For information about the possible destination types, see the ***ArcSight SmartConnector User Guide***.

FlexConnectors are custom connectors you define to gather security events from log files, databases, and other software and devices. FlexConnectors let you create custom connectors that can read and parse information from third-party devices and map that information to ArcSight's event schema.

FlexConnector Development

This guide describes these basic steps for creating a FlexConnector:

- Deciding the type of FlexConnector to develop based on the source data to be collected. (See "Choose a FlexConnector Type".)
- Providing a log file. For database connectors, this includes developing a query for pulling events.
- Installing and configuring one of the FlexConnector types. For SNMP, you install the SNMP Unified connector; for syslog, you install the Syslog Daemon connector. (See "Install and Configure theFlexConnector".)
- Creating your properties file (parser) and mapping events to ArcSight fields. (See "Create a Configuration File".)
- Creating the categorization.csv file and assigning appropriate categories. (See "FlexConnectors and Categorization".)

This guide also describes related topics, such as:

- Log Rotation Types
- Unparsed Events Detection
- Advanced Parameters that can be used to tune the collection process
- Map Files
- ArcSight Operations
- FlexConnectors and Categorization
- Configuring a Connector for ArcSight ESM Domain Field Sets

IPv6-Aware Parsers

With only a few exceptions, ArcSight SmartConnectors fully support IPv6 and IPv4 addresses for event receiving and event processing. An extra mapping used to be required to map IPv6 addresses. In case the destination is not an IPv6-Aware destination, the IP Address is automatically mapped to the corresponding Device Custom IPv6 Address fields.

Note Older versions of FlexConnector and parsers continue to use Device Custom IPv6 Address fields for IPv6 addresses.

Event Fields

The following ArcSight event fields accept both IPv4 and IPv6 addresses, in case the destination is an IPv6-Aware destination. For more information, see ["ArcSight Built-in Event Field Mappings" on page 190](#).

- Destination Address
- Destination Translated Address
- Device Address
- Device Translated Address
- Source Address
- Source Translated Address

The Bytes In and Bytes Out event field parameters have been changed to Long Data Type.

For IPv6-aware parsers, the Device Custom IPv6 address 1, 2, and 3 fields can contain either IPv4 or IPv6 addresses. These fields are rarely used, if so, the corresponding labels should be set to with an appropriate value.

Operations

The behavior of the following operations has been changed to support IPv6-aware parsers,. For more information, see ["ArcSight Operations" on page 173](#).

- __byteArrayToIPAddress (new parameter)
- __byteArrayToIPv6
- __getIPv4AddressEmbeddedInIPv6Address
- __hexStringToAddress
- __hexStringToIPv6Address
- __oneOfAddress
- __oneOfHostName
 - __stringToIPv6Address

Developer Considerations

The main IPv6-aware parser and all of the SmartConnector extra processors should be marked with the property **ipv6.aware=true**.

Mixed ranges are not supported (that being, where one end of the ranges is an IPv4 address and the other is an IPv6 address).

Since the standard IP address fields (such as Device Address, Source Address, Destination Address, and so on) support both IPv4 and IPv6 addresses, the Device Custom IPv6 Address fields are rarely required in an IPv6-aware parser. These mappings should be redirected to the standard address fields. If there are any addresses which do not fall into any of the normal device, source, or destination categories, then the Device Custom IPv6 Address fields can be used, but that would be a very rare case.

Do not use the `__stringToIPv6Address` or `__byteArrayToIPv6` operations as they are not relevant in IPv6-aware parsers.

Rename the `__byteArrayToIPv6` operation to the new generic `__byteArrayToIPAddress` operation in the parser.

- The `__oneOfAddress` operation returns the first non-null IP address whether an IPv4 or an IPv6, when that operation is used in an IPv6-aware parser.
- The Bytes In and Bytes Out event fields are now Long Data Type.

Folder Structure

The following table lists the connector folder structure after connector installation and configuration, and describes the contents of each folder.

Directory	Description
\$ARCSIGHT_HOME\current\bin	Executables and scripts; for example, runagentsetup.bat .
\$ARCSIGHT_HOME\current\config\agent	Default and base configurations; for example, agent.defaults.properties .
\$ARCSIGHT_HOME\current\logs	Generated logs; for example agent.log .
\$ARCSIGHT_HOME\current\user\agent	Connector property files and destination-specific configurations; for example, agent.properties .
\$ARCSIGHT_HOME\current\user\agent\agentdata	Queue, cache, and persistence files.
\$ARCSIGHT_HOME\current\user\agent\acp\categorizer\current	Categorization files (ArcSight Content), which provide additional meaning to events.
\$ARCSIGHT_HOME\current\user\agent\flexagent	Custom parsers that are developed for the FlexConnector.
\$ARCSIGHT_HOME\current\user\agent\map	Mapping files that can be used to set fields in the Security Event object; for example, map.0.properties .

Key Files

During connector installation and configuration, several key files are created. The following table describes these files, their locations, and their purpose.

File Name	Path	Description
agent.log	\$ARCSIGHT_HOME\current\logs	Generated log that contains information on the running of the connector; search for ERROR to see any errors that occurred during the running of the connector. The most current log is agent.log , but there can be older logs in the folder as well, such as agent.log.1 or agent.log.2 .
agent.properties	\$ARCSIGHT_HOME\current\user\agent	Contains configuration parameters and values, created from the values entered during connector configuration.
agent.default.properties	\$ARCSIGHT_HOME\current\config\agent	Contains default framework parameters; for example, contains the syntax for enabling debugging and increasing the agent.log file size and agent log count. Do not modify agent.default.properties as it is overwritten when the connector is upgraded. Make any property changes in agent.properties .

FlexConnector Management

There are currently two ways to manage SmartConnectors: through ArcSight Management Center (also referred to as "ArcMC") and through ArcSight Connector Appliance. Eventually, ArcSight Management Center will replace ArcSight Connector Appliance.

ArcSight Connector Appliance

The ArcSight Connector Appliance is a hardware solution that incorporates a number of onboard ArcSight SmartConnectors and a web-based user interface that provides centralized management for SmartConnectors across a potentially large number of hosts.

FlexConnectors can generally be managed by a Connector Appliance and can be hosted on the appliance if they are compatible with a Linux platform. The Connector Appliance ships with several prototype FlexConnectors, including the following:

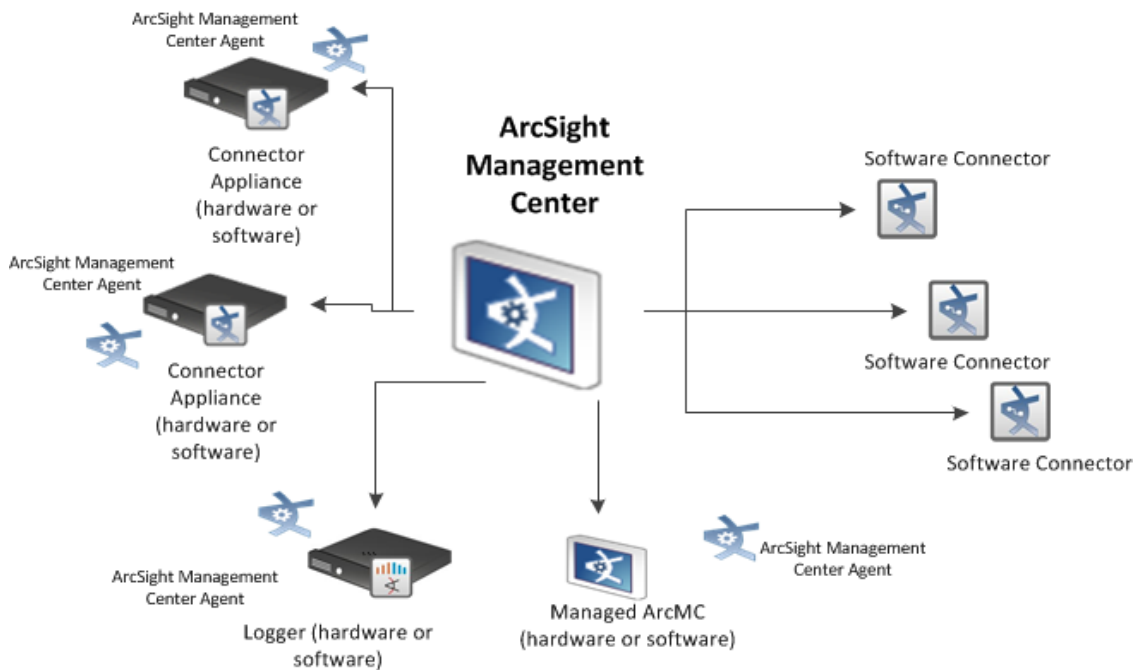
- ArcSight FlexConnector File
- ArcSight FlexConnector ID-Based DB
- ArcSight FlexConnector Multiple DB
- ArcSight FlexConnector Regex File

- ArcSight FlexConnector Regex Folder File
- ArcSight FlexConnector Simple Network Management Protocol (SNMP Unified)
- ArcSight FlexConnector Time-Based DB
- ArcSight FlexConnector XML File

For detailed information and instructions for using the Connector Appliance, see the *ArcSight Connector Appliance Administrator's Guide*.

ArcSight Management Center

ArcSight Management Center includes all of the functions of ArcSight Connector Appliances, and also the ability to manage and monitor an additional range of ArcSight products, such as Connector Appliances, Loggers, and other ArcSight Management Centers, as illustrated in the following figure.



ArcSight Management Center uses the concept of nodes to manage various entities. A node is a networked ArcSight product that can be centrally managed using ArcSight Management Center. Each node is associated with a single networked host that has been assigned either a hostname, an IP address, or both.

A single host can include multiple nodes. For example, a single Connector Appliance (with a single IP address or hostname) could have multiple containers, each of which could be a separate node. In addition, a node can be in a parent or child relationship with other nodes.

You can perform any of the following node management tasks:

- View managed nodes by location, host, or node type
- Add, view, edit, and delete locations for hosts

- Add nodes from a host, import hosts from a **.csv** file, view and delete hosts, view all hosts in a location, move hosts to different locations, and scan hosts for new connectors or containers

See the *ArcSight Management Center Administrator's Guide* for details.

Chapter 2: Choose a FlexConnector Type

The FlexConnector type you choose should be based on the format of the security event data. Examples of data formats for different FlexConnector types are provided in [“Event Data Format Examples”](#).

FlexConnector Types

The available FlexConnector types are listed in the following table; selection criteria is included.

FlexConnector Type	Description
File	<p>Choose this type if the event data is in log files that use a fixed, delimited format. In this case, each line in the text file represents a unique event, and each line contains the same number of fields, in the same order. Fixed-format log files can be delimited by commas, tabs, or another character, such as a pipe (' ').</p> <p>All file-reader FlexConnectors can process GZIP and ZIP files. Other compression formats are not supported. Compressed files are processed in batch mode only. The connectors read the file from the beginning to the end and then stop monitoring the file. See "Log File FlexConnector" and "ArcSight FlexConnector File".</p>
ID-Based Database	<p>Choose ID-Based Database or Time-Based Database for devices that write security event information to a database. Each row represents a single event, and the number and meaning of the columns are fixed. If you use unique IDs to read events from a database, choose ID-Based Database.</p> <p>Knowledge of SQL is a prerequisite for coding database FlexConnectors. See "ID-Based Database FlexConnector" and "ArcSight FlexConnector ID-Based Database".</p>
JSON Folder Follower	<p>Choose this type for devices that write event information to JSON files. Event information in these files is presented in standard JSON format. This type recursively reads events from JSON-based files in a folder. See "JSON Folder Follower FlexConnector/JSON Multiple Folder Follower FlexConnector" and "ArcSight FlexConnector JSON Multiple Folder Follower".</p>
Multiple Database	<p>Choose this type to retrieve information from multiple databases that use the same query or retrieve different set of events using different queries from the same database.</p> <p>Knowledge of SQL is a prerequisite for coding database FlexConnectors. See "Multiple Database FlexConnector" and "ArcSight FlexConnector Multiple Database".</p>
Multiple Folder File	<p>Choose this type for devices that write log files to multiple folders. This connector type can read events in real time or in batch mode. See "Multiple Database FlexConnector" and "ArcSight FlexConnector Multiple Folder File".</p>

FlexConnector Type	Description
Regex File	<p>Choose this type if the source log files have one event per line, but the format of each line varies based on the type of event information. In this case, each line shares a common section (for example, the date and hostname), but the number and content of the other fields on the line varies.</p> <p>The regular expression-based FlexConnectors require a familiarity with Java-compatible regular expressions. See "Regex FlexConnectors (Variable-Format File FlexConnectors)" and "ArcSight FlexConnector Regex File".</p>
Regex Folder File	<p>File and Regex File FlexConnectors read events in real time, one line at a time, from a log file. However, some devices may not write to log files in real time. To read such events, use a Regex Folder Follower FlexConnector. This connector processes all log files in a specified folder.</p> <p>The regular expression-based FlexConnectors require a familiarity with Java-compatible regular expressions. See "Regex FlexConnectors (Variable-Format File FlexConnectors)" and "ArcSight FlexConnector Regex Folder File".</p>
REST	<p>The REST FlexConnector uses REST API endpoints, JSON parser, and OAuth2 authentication to collect security events from cloud vendors (such as Salesforce or Google Apps). See "ArcSight FlexConnector REST". For detailed information about this FlexConnector, see the <i>ArcSight REST FlexConnector Developer's Guide</i> for details.</p>
Scanner DB Scanner Text Reports Scanner XML Reports	<p>Choose a Scanner FlexConnector type to import the results of a scan from a scanner device and forward the data to ESM so that ESM can model an organization's assets, open ports, operating systems, applications, and vulnerabilities. The connector imports periodic scans to ESM, which uses this information for event prioritization, reporting, and correlation.</p> <p>Database:</p> <p>A database contains results for multiple scans where each scan is identified by a job identifier (ID). The scan results are organized in multiple tables that are linked by job IDs or other IDs. SQL query-based parsers are used to extract relevant information from the scan results.</p> <p>Knowledge of SQL is a prerequisite for coding database FlexConnectors. See "Scanner FlexConnector" and "ArcSight FlexConnector Scanner Database".</p> <p>Text Reports:</p> <p>A normal text report contains results for a single scan with each line in the report containing a piece of information about a host. Regular expression based parsers are used to extract relevant information from the report.</p> <p>The regular expression-based FlexConnectors require a familiarity with Java-compatible regular expressions. See "Scanner FlexConnector" and "ArcSight FlexConnector Scanner Text Reports".</p> <p>XML Reports:</p> <p>An XML report contains results for a single scan with scan results organized in the form of nested XML elements. XQuery/XPath-based parsers are used to extract relevant information from the report.</p> <p>The XML FlexConnector require a familiarity with XML, XPath, and XQuery. See "Scanner FlexConnector" and "ArcSight FlexConnector Scanner XML Reports".</p>

FlexConnector Type	Description
Time-Based Database	<p>Choose ID-Based Database or Time-Based Database for devices that write security event information to a database. Each row represents a single event, and the number and meaning of the columns are fixed. One column represents the event timestamp and can be used to order the rows. To read events from database table rows, choose Time-Based DB.</p> <p>Knowledge of SQL is a prerequisite for coding database FlexConnectors. See "Time-Based Database FlexConnector" and "ArcSight FlexConnector Time-Based DB".</p>
XML File	<p>Choose this type for devices that write event information to XML files. Event information in these files is presented in standard XML format, using namespaces, elements, attributes, text, and cdata. This connector type recursively reads events from XML-based files in a folder.</p> <p>The XML FlexConnectors require a familiarity with XML, XPath, and XQuery.</p> <p>See "XML File FlexConnector" and "ArcSight FlexConnector XML File". See "Developing an XML FlexConnector" for a description of the development of an example of an XML FlexConnector.</p>
SNMP Unified	<p>For SNMP devices, choose the SmartConnector for SNMP Unified. See the SmartConnector configuration guide for installation and configuration information. See "SNMP FlexConnector" and "ArcSight FlexConnector Simple Network Management Protocol (SNMP Unified)".</p>
Syslog	<p>For reading events from syslog messages, choose the SmartConnector for Syslog Daemon and define a Syslog FlexConnector sub-connector to parse syslog packets of interest. See "Syslog FlexConnector" and "ArcSight FlexConnector Syslog".</p>

Event Data Format Examples

You choose a FlexConnector type based on the format of the event data. The following examples illustrate the kind of source data expected by the various FlexConnector types.

Log File FlexConnector

The following is an example of a fixed-format, delimited log file. In this example, there are three events; each has the same format composed of six tokens separated by a comma.

```
01/01/2016-11:33:00,1.1.1.1,52123,2.2.2.2,80,Invalid URL
01/01/2016-12:43:00,3.3.3.3,49123,2.2.2.2,80,Buffer Overflow Attempt
01/01/2016-13:53:00,4.4.4.4,35123,2.2.2.2,80,Web Cgi Access
```

ID-Based Database FlexConnector

Two rows of a security event table in a database might look like this. This example describes two events: one with ID 123456 and another with ID 123457.

EventId	Incident Time	Signature	SourceIP	Destination IP	Priority	Protocol
123456	09/01/16 12:56:00	Port Scan	9.10.11.12	13.14.15.16	1	TCP
123457	09/01/16 12:54:00	ICMP Failure	1.2.3.4	5.6.7.8	3	ICMP

JSON Folder Follower FlexConnector/JSON Multiple Folder Follower FlexConnector

An JSON file with event information might look like this:

```
{
  "chunk_size":100,
  "entries":[{
    "source":null,
    "created_by":{
      "type":"user",
      "id":"175265599",
      "name":"Mary Jane",
      "login":"mary.jane@abc.com"
    },
    "created_at":"1324497497",
    "event_id":"13254621",
    "event_type":"FAILED_LOGIN",
    "ip_address":"192.168.233.76",
    "type":"event",
    "session_id":null
  },
  {
    "source":null,
    "created_by":{
      "type":"user",
      "id":"175265599",
      "name":"Mary Jane",
      "login":"mary.jane@abc.com"
    },
    "created_at":"1324497544",
    "event_id":"13254633",
    "event_type":"FAILED_LOGIN",
    "ip_address":"192.168.233.76",
    "type":"event",
```

```
        "session_id":null
    },
    {
        "source":null,
        "created_by":{
            "type":"user",
            "id":"175265599",
            "name":"Mary Jane",
            "login":"mary.jane@abc.com"
        },
        "created_at":"1324497614",
        "event_id":"13254649",
        "event_type":"LOGIN",
        "ip_address":"192.168.233.76",
        "type":"event",
        "session_id":null
    }
  ]
}
```

Multiple Database FlexConnector

The Multi-Database FlexConnector reads events from more than one database or multiple event types from different tables in the same database. For data format examples, see ["ID-Based Database FlexConnector"](#) and ["Time-Based Database FlexConnector"](#).

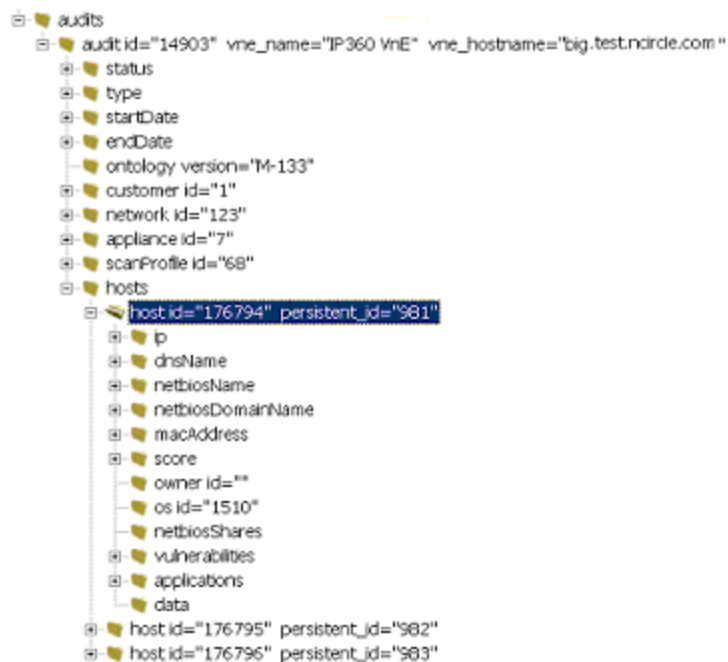
Regex FlexConnectors (Variable-Format File FlexConnectors)

FlexConnectors , capable of processing variable-format log files, include Regex Log File, Regex Folder Follower and Regex Multiple Folder Follower. Variable-format log files might look like this:

```
Aug 21 15:28:49 beach sshd[24939]: Failed password for rajiv from
192.168.10.27 port 33654 ssh2
Aug 21 15:28:51 beach sshd[24939]: Accepted password for rajiv from
192.168.10.27 port 33654 ssh2
Aug 21 15:28:51 beach PAM_unix[24948]: (ssh) session opened for user rajiv by
(uid=525)
Aug 21 15:28:53 beach PAM_unix[24948]: (ssh) session closed for user rajiv
Aug 22 00:13:23 beach sshd[6305]: Did not receive IDentification string from
192.168.10.28
```


Scanner FlexConnector

The following is an example scan report:



SNMP FlexConnector

SNMP traps contain variables (varbinds) that must be mapped to the ArcSight Database Schema. The SmartConnector for SNMP Unified supports SNMP traps in versions 1, 2, and 3. The following example is the output of an SNMP connector when it receives a trap (in this case, generated by SecureNet Pro) for which it is not yet configured:

```
[Wed May 21 11:11:17 PDT 2016] [INFO ] Unable to process trap (not configured) :
```

```
Received SNMPv1 trap
```

```
Port : 162
Generating Agent : 10.0.112.104
Sending Agent : 10.0.112.104
Time Stamp : 412257333
Enterprise OID : 1.3.6.1.4.1.8678.1.1.2
Trap Type : 1
Var Binds:14
```

```
VarBind #0
```

```
0.0.0.0.0.0.0.0.412257333.0
StringValue: 439228089
```

```
    TimeStamp: 0
    Value: 439228089
VarBind #1
    0.0.0.0.0.0.0.0.412257333.0
    StringValue: 439228089
    TimeStamp: 0
    Type: ASN_INTEGER | ASN_INTEGER32
    Value: 439228089
VarBind #2
    0.0.0.0.0.0.0.0.412257333.0
    StringValue: [] - TCP Connection from 10.0.112.132
    TimeStamp: 0
    Type: ASN_OCTSTR
    Value: [B@29e357
VarBind #3
    0.0.0.0.0.0.0.0.412257333.0
    StringValue: TCP Session Logging
    TimeStamp: 0
    Type: ASN_OCTSTR
    Value: [B@ca470
VarBind #4
    0.0.0.0.0.0.0.0.412257333.0
    StringValue: Miscellaneous
    TimeStamp: 0
    Type: ASN_OCTSTR
    Value: [B@7fc686
VarBind #5
    0.0.0.0.0.0.0.0.412257333.0
    StringValue: TCP (Stream)
    TimeStamp: 0
    Type: ASN_OCTSTR
    Value: [B@42bece
VarBind #6
    0.0.0.0.0.0.0.0.412257333.0
    StringValue: 1
    TimeStamp: 0
    Type: ASN_INTEGER | ASN_INTEGER32
    Value: 1
    VarBind #7
    0.0.0.0.0.0.0.0.412257333.0
    StringValue: 05/21/2003 10:58:26
    TimeStamp: 0
```

```
    Type: ASN_OCTSTR
    Value: [B@7cfa52
VarBind #8
    0.0.0.0.0.0.0.0.412257333.0
    StringValue: 00:b0:d0:61:6c:6e
    TimeStamp: 0
    Type: ASN_OCTSTR
    Value: [B@161dff
VarBind #9
    0.0.0.0.0.0.0.0.412257333.0
    StringValue: 00:00:d1:ee:c4:2e
    TimeStamp: 0
    Type: ASN_OCTSTR
    Value: [B@b81e3
VarBind #10
    0.0.0.0.0.0.0.0.412257333.0
    StringValue: 10.0.112.132
    TimeStamp: 0
    Type: ASN_OCTSTR
    Value: [B@7c6e42
VarBind #11
    0.0.0.0.0.0.0.0.412257333.0
    StringValue: 10.0.111.26
    TimeStamp: 0
    Type: ASN_OCTSTR
    Value: [B@2af0b3
VarBind #12
    0.0.0.0.0.0.0.0.412257333.0
    StringValue: 60901
    TimeStamp: 0
    Type: ASN_OCTSTR
    Value: [B@2082e2
VarBind #13
    0.0.0.0.0.0.0.0.412257333.0
    StringValue: 64288
    TimeStamp: 0
    Type: ASN_OCTSTR
    Value: [B@70c85e
```

Syslog FlexConnector

A security appliance might send syslog messages with the following format:

Myapplication: Intruder Detected from 1.1.1.1 to 2.2.2.2 High

In this case, **Myapplication** is the name of the security appliance, **Intruder Detected** is the name of the event, **1.1.1.1** and **2.2.2.2** are the source and target addresses and **High** refers to the severity of the event. This message is not delimited; however, you can identify that this message comes from the security appliance by the prefix **Myapplication**. Regular expressions are a simple mechanism to identify and tokenize the message, so the format of a FlexConnector Syslog configuration file is similar to the FlexConnector Regex Log-file. The only difference is that the detected time and sending host will automatically be set by the syslog daemon and only additional mappings need to be specified.

Time-Based Database FlexConnector

Two rows of a security event table in a database might look like this. This example describes two events: one at 12:56 and another at 12:54.

EventId	Incident Time	Signature	SourceIP	Destination IP	Priority	Protocol
CCC-DDD	09/01/16 12:56:00	Port Scan	9.10.11.12	13.14.15.16	1	TCP
AAA-BBB	09/01/16 12:54:00	ICMP Failure	1.2.3.4	5.6.7.8	3	ICMP

XML File FlexConnector

An XML file with event information looks like this:

```
<?xml version="1.0" encoding="UTF-8" ?>
-   <mycompanyReport version="1.1">
-   <reportHeader>
        <copyrightNotice value="Copyright 2016 MyCompany, Inc." />
        <trademarkNotice value="MyCompany is a registered trademark of MyCompany,
Inc. All rights reserved." />
        <productVersion value="MyCompany for Servers version 1.2.3 for Windows(R)
Operating Systems" />
        <reportFile value="Memory Mapped File" />
        <reportFileEncrypted value="0" />
        <policyFile value="C:\Program Files\MyCompany\MyCompany Trial
Kit\SMTP\policy\mc.pol" />
        <configFile value="C:\Program Files\MyCompany\MyCompany Trial
Kit\SMTP\bin\mc.cfg" />
        <databaseFile value="C:\Program Files\MyCompany\MyCompany Trial
Kit\SMTP\db\Application.twd" />
```

```

    <systemName value="HOGWARTS" />
    <commandLine value="C:\Program Files\MyCompany\MyCompany Trial
Kit\SMTP\bin\MyCompany.exe --check --no-tty-output --cfgfile C:\Program
Files\MyCompany\MyCompany Trial Kit\SMTP\bin\mc.cfg -- email-report --email-
report-level 3 --report-format xml --twrfile
    C:\Program Files\MyCompany\MyCompany Trial Kit\SMTP\report\Report- .twr" />
    <ipAddress value="172.16.252.58" />
    <creator value="SYSTEM" />
    <hostID value="S-1-5-21-3494633144-188423603-1740787705" />
    <creationTime raw="1117725227" value="Thu, 02 Jun 2005 10:13:47 -0500" />
    <lastDBUpdateTime raw="0" value="Never" />
  </reportHeader>
-   <section type="NTFS" name="Windows File System">
-   <rule name="SMTP Server" startPoint="C:\Program Files\MyCompany\MyCompany
Trial Kit\active_files\SMTP\bin\help">
-   <ruleHeader>
    <severity value="30" />
    <onViolation value="" />
    <match value="" />
    <emailAddressList />
  </ruleHeader>
-   <ruleSummary>
    <violationCount value="3" />
    <addedCount value="0" />
    <removedCount value="3" />
    <changedCount value="0" />
  </ruleSummary>
    <errorList />
-   <added>
    <object name="C:\Program Files\MyCompany\MyCompany Trial Kit\active_
files\SMTP\bin\help\smtpserver.pdf" />
    </added>
-   <removed>
    <object name="C:\Program Files\MyCompany\MyCompany Trial Kit\active_
files\SMTP\bin\help\releasenotes.txt" />
    </removed>
-   <changed>
    <object name="C:\Program Files\MyCompany\MyCompany Trial Kit\active_
files\SMTP\bin\help\xyz.txt" />
    </changed>
  </rule>
-   <sectionSummary>

```

```
<objectsScanned value="35" />  
<sectionViolationCount value="26" />  
<sectionMaxSeverity value="100" />  
</sectionSummary>  
</section>  
</mycompanyReport>
```

Chapter 3: Install and Configure the FlexConnector

Installation and configuration consists of installing the FlexConnector core software, and then selecting and configuring the destination for the log messages.

FlexConnector Installation

The installation process installs the framework, tools, and sample files necessary for configuring a FlexConnector. Once a FlexConnector is installed, it functions the same as any SmartConnector.

The installation directory (for example, **C:\FlexConnector\current**) is referred to as **\$ARCSIGHT_HOME**, regardless of the platform.

To successfully configure a FlexConnector, the ArcSight Manager or Logger and database components with which the FlexConnector will communicate must be up and running. The FlexConnector tries to connect to the destination during the configuration process. If it cannot connect, configuration fails.

Install Core Software

A FlexConnector can be installed on all ArcSight supported platforms; for the complete list, see the **SmartConnector Platform Support** document, available from the Micro Focus SSO and Protect 724 sites.

1. Download the SmartConnector executable for your operating system from the Micro Focus SSO site.
2. Start the SmartConnector Installer by running the executable for your operating system platform.

Follow the installation wizard through the following folder selection tasks and installation of the core connector software:

Introduction

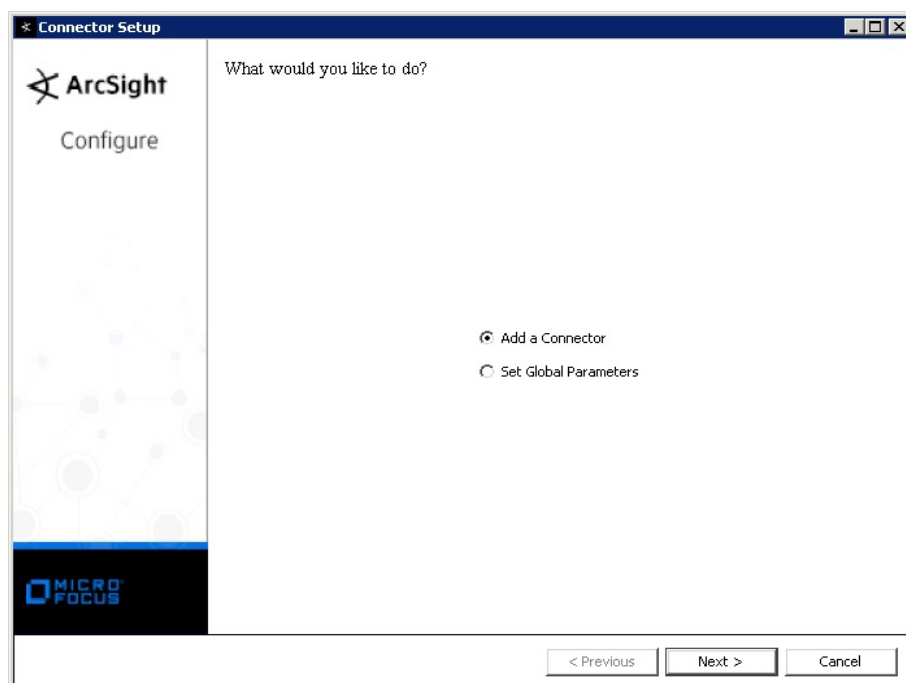
Choose Install Folder

Choose Shortcut Folder

Pre-Installation Summary

Installing...

3. When the installation of SmartConnector core component software is finished, the following window is displayed:



Set Global Parameters (Optional)

If you choose to perform any of the operations shown in the following table, do so before adding your connector. After installing core software, you can set the following parameters:

Global Parameter	Setting
FIPS mode	Set to Enabled to enable FIPS compliant mode. To enable FIPS Suite B Mode, see the <i>SmartConnector User Guide</i> under "Modifying Connector Parameters" for instructions. Initially, this value is set to Disabled .
Remote Management	Set to Enabled to enable remote management from ArcSight Management Center. When queried by the remote management device, the values you specify here for enabling remote management and the port number will be used. Initially, this value is set to Disabled .
Remote Management Listener Port	The remote management device will listen to the port specified in this field. The default port number is 9001.
Preferred IP Version	When both IPv4 and IPv6 IP addresses are available for the local host (the machine on which the connector is installed), you can choose which version is preferred. Otherwise, you will see only one selection. The initial setting is IPv4 .

The following parameters should be configured only if you are using Micro Focus SecureData solutions to provide encryption. See the *Micro Focus SecureData Architecture Guide* for more information.

Global Parameter	Setting
Format Preserving Encryption	Data leaving the connector machine to a specified destination can be encrypted by selecting 'Enabled' to encrypt the fields identified in 'Event Fields to Encrypt' before forwarding events. If encryption is enabled, it cannot be disabled. Changing any of the encryption parameters again will require a fresh installation of the connector.
Format Preserving Host URL	Enter the URL where the Micro Focus SecureData server is installed.
Proxy Server (https)	Enter the proxy host for https connection if any proxy is enabled for this machine.
Proxy Port	Enter the proxy port for https connection if any proxy is enabled for this machine.
Format Preserving Identity	The Micro Focus SecureData client software allows client applications to protect and access data based on key names. This key name is referred to as the identity. Enter the user identity configured for Micro Focus SecureData.
Format Preserving Secret	Enter the secret configured for Micro Focus SecureData to use for authentication.
Event Fields to Encrypt	Recommended fields for encryption are listed; delete any fields you do not want encrypted from the list, and add any string or numeric fields you wish to be encrypted. Encrypting more fields can affect performance, with 20 fields being the maximum recommended. Also, because encryption changes the value, rules or categorization could also be affected. Once encryption is enabled, the list of event fields cannot be edited.

After making your selections, click **Next**. A summary screen is displayed. Review the summary of your selections and click **Next**. Click **Continue** to return to the "Add a Connector" window. Continue the installation procedure with "[Select Connector and Add Parameter Information](#)."

Select Connector and Add Parameter Information

1. Select **Add a Connector** and click **Next**. If applicable, you can enable FIPS mode and enable remote management later in the wizard after connector configuration.
2. Select a specific connector to install. The FlexConnectors are mostly grouped together beginning with ArcSight FlexConnector. The exceptions are syslog FlexConnectors (choose **Syslog Daemon**) and SNMP FlexConnectors (choose **SNMP Unified** connector). Click **Next** when you have made your selection.
3. Enter the required SmartConnector parameters to configure the SmartConnector, then click **Next**.
The installation wizard prompts for different parameters depending upon the type of FlexConnector or Syslog SmartConnector selected. In addition to the parameters you can configure through the installation wizard, you can also configure parameters directly in the **agent.properties** file. Those parameters are discussed in "[Advanced Parameters](#)".

ArcSight FlexConnector File

Choose this type if the event data is in log files that use a fixed, delimited format. In this case, each line in the text file represents a unique event, and each line contains the same number of fields, in the same order.

Fixed-format log files can be delimited by commas, tabs, or another character, such as a pipe ('|').

Parameter	Description
Log Unparsed Events	The default value is false . Select true for the connector to detect and log unparsed events to \$ARCSIGHT_HOME\current\logs\events.log . For more information on unparsed events, see " Unparsed Events Detection ".
Log File Name	The absolute path and name of the file that this FlexConnector will read. For example: c:\temp\sample_data.txt
Configuration File	<p>The base name of the configuration file that describes the format of the log file. For a connector that parses fixed-format files, the suffix .sdkfilereader.properties is appended automatically.</p> <p>For a connector that parses variable-format files, the suffix .sdrfilereader.properties is appended automatically.</p> <p>For example, if you specify the following name for a configuration file that parses fixed-format log files: sample</p> <p>The filename becomes:</p> <p>ARCSIGHT_HOME\user\agent\flexagent\sample.sdkfilereader.properties.</p>

ArcSight FlexConnector ID-Based Database

Choose this type for devices that write security event information to a database. This type will read events from the database based on unique IDs. (If the connector is to read events from database table rows, you should select [ArcSight FlexConnector Time-Based DB](#).)

Note:

- After installing connector core software and before configuring the ArcSight FlexConnector ID-Based DB, you will need to download an appropriate JDBC driver. See [“Additional Configuration for Database Connectors”](#) for complete information.
- Knowledge of SQL is a prerequisite for coding database FlexConnectors.
- SmartConnector releases since 7.2.1 have implemented Java 8, which does not support ODBC connections; therefore, database connectors can only use JDBC connections. For the same reason, the MS Access database, which uses only ODBC connections, is no longer supported.

Connector Setup

ArcSight
Configure

Enter the parameter details

Database JDBC Driver: com.microsoft.sqlserver.jdbc.SQLServerDriver

Database URL:

Database User:

Database Password:

Configuration Folder:

Query Frequency: 5

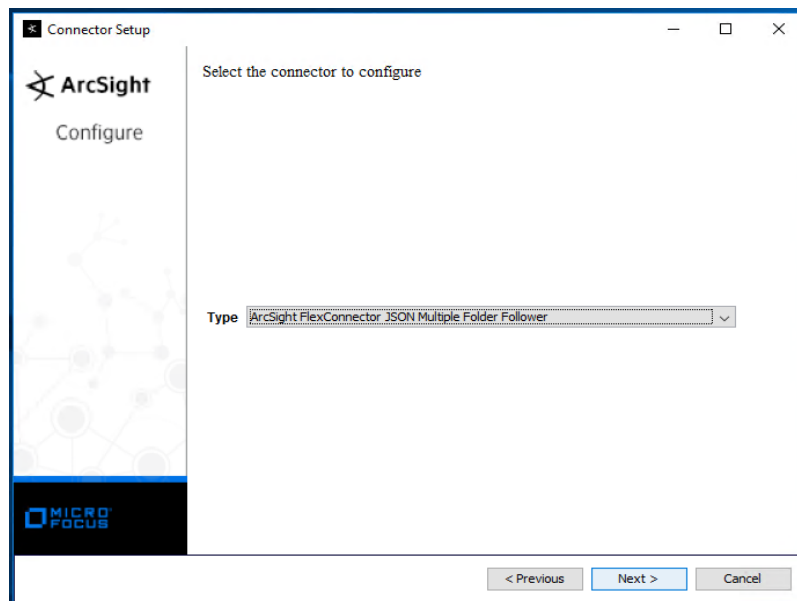
< Previous Next > Cancel

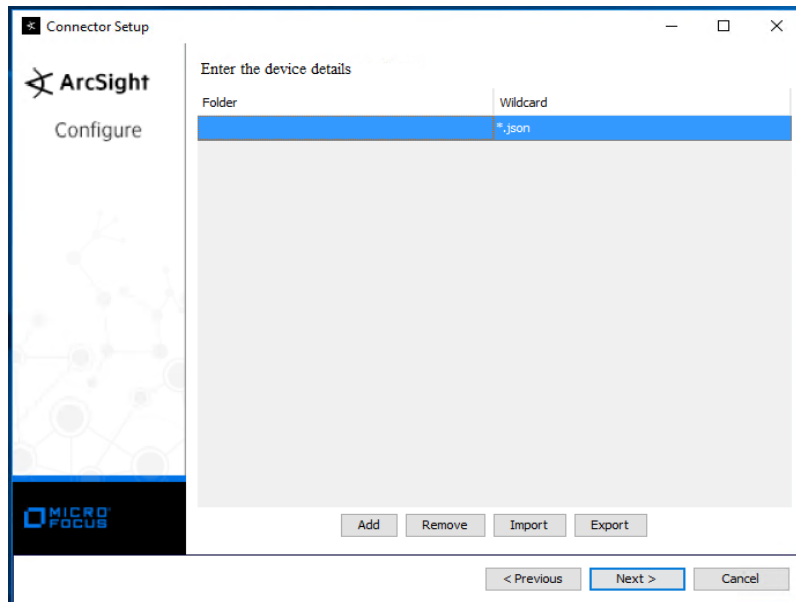
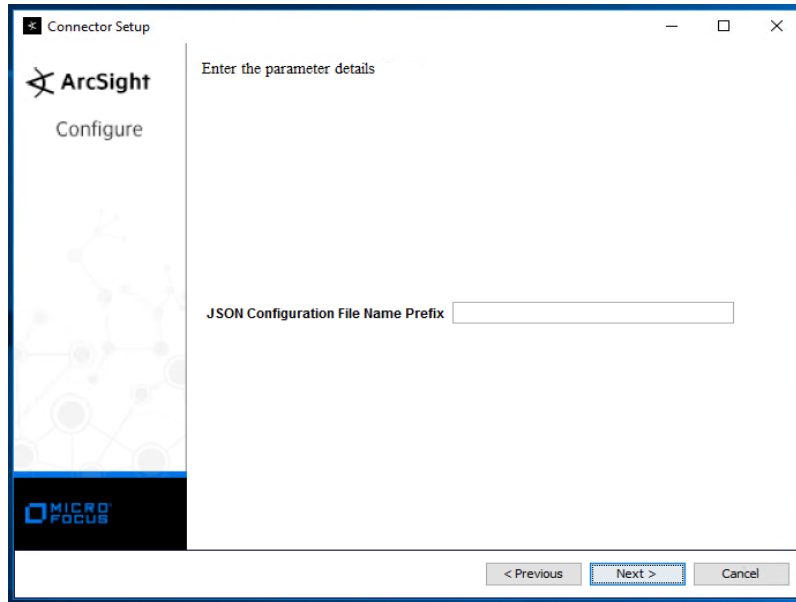
Parameter	Description
Database JDBC Driver	<p>The JDBC driver that will be used to connect to the database.</p> <ul style="list-style-type: none"> For SQL Server, perform the procedure in "Install SQL Server JDBC Driver". For MySQL, perform the procedure in "Install MySQL Driver". For Oracle, use: oracle.jdbc.driver.OracleDriver This default Oracle JDBC driver works with Oracle 9i, 10g, 11g, and 12c database versions. If you are using Oracle 8i, see "Oracle 8i Support". For PostgreSQL, use: org.postgresql.Driver For DB2 unified driver, use: com.ibm.db2.jcc.DB2Driver For DB2 Legacy CLI-based, use: COM.ibm.db2.jdbc.net.DB2Driver For Sybase, use: com.sybase.jdbc2.jdbc.SybDriver
Database URL	<p>The JDBC URL that identifies the database.</p> <ul style="list-style-type: none"> For Oracle, use: jdbc:oracle:thin:@hostname_or_IP:1521:database_name For MySQL, use: jdbc:mysql://hostname_or_IP:3306/database_name For Microsoft SQL Server 2000, use: jdbc:microsoft:sqlserver://host:port;databasename=name For Microsoft SQL Server 2005 and later, use: jdbc:sqlserver://host:port;databasename=name For PostgreSQL, use: jdbc:postgresql://host/database For DB2 unified driver, use: jdbc:db2:database_name For DB2 Legacy CLI-based, use: jdbc:db2://host_name: port_number/ database_name For Sybase, use: jdbc:sybase:Tds: hostname_or_IP:5000/sybsecurity
Database User	The database user name.

Parameter	Description
Database Password	Password for the database user.
Configuration Folder	<p>Enter the name of the folder that contains the properties file. Do not enter the full path to the file as doing so will result in an error.</p> <p>This is also the root name of the configuration file. If the configuration folder is myfolder, then the FlexConnector will look for the configuration file in the directory: ARCSIGHT_HOME\user\agent\flexagent\myfolder</p> <ul style="list-style-type: none">• The configuration file for time-based connectors will be named: myfolder.sdktdatabase.properties• The configuration file for ID-based connectors will be named: myfolder.sdkidatabase.properties
Query Frequency	Specifies how often, in seconds, to query the database. The default is 5 seconds.

ArcSight FlexConnector JSON Multiple Folder Follower

Choose this type for devices that write event information to JSON files. Event information in these files is presented in standard JSON format. This connector recursively reads events from JSON-based files in multiple folders



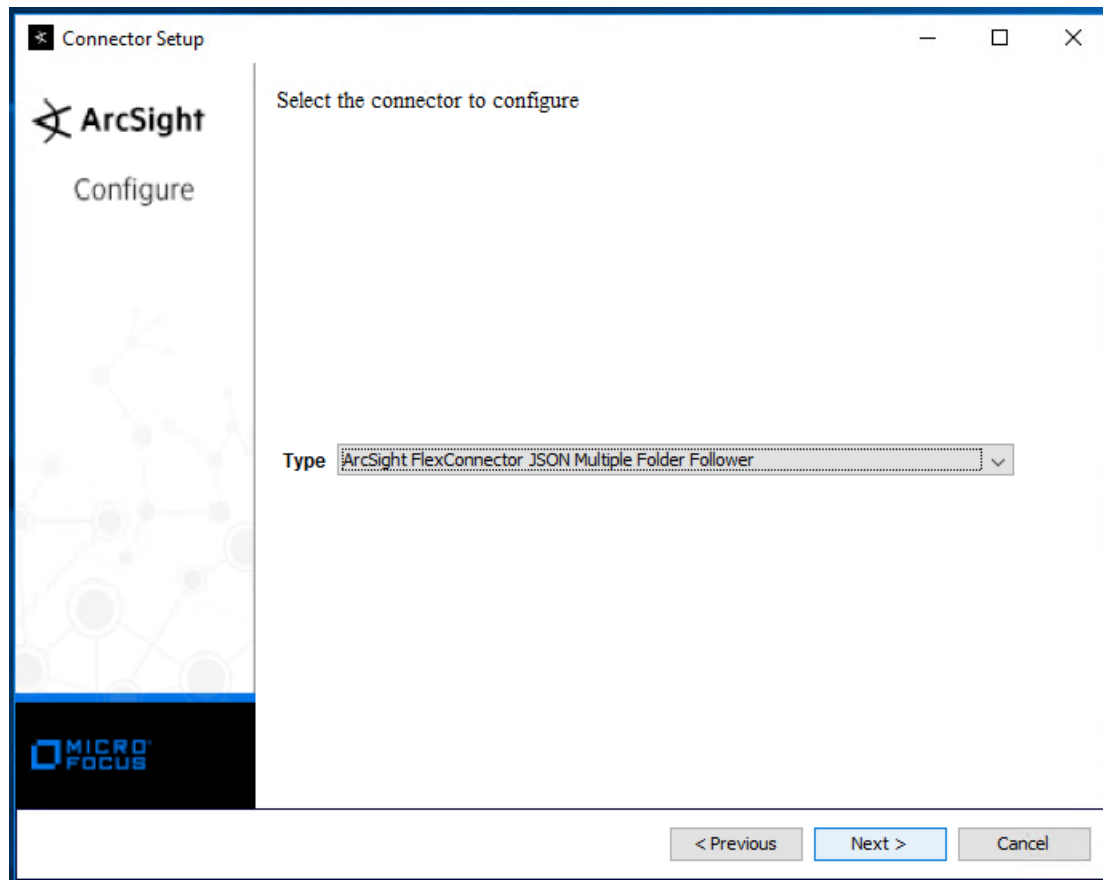


Parameter	Description
JSON Configuration File Name Prefix	<p>The base name of the configuration file that describes the format of the log file.</p> <p>The suffix <code>.jsonparser.properties</code> is appended automatically. For example, if you specify:</p> <p><code>vendor_product</code></p> <p>The filename becomes:</p> <p><code>\$ARCSIGHT_HOME\user\agent\flexagent\vendor_product.jsonparser.properties</code></p>
Folder	<p>The absolute path of the directory where log files for the FlexConnector are located. For example: <code>c:\logs</code></p>
Wildcard	<p>Enter a Wildcard that identifies the files to process. The default wildcard is <code>*.json</code></p>

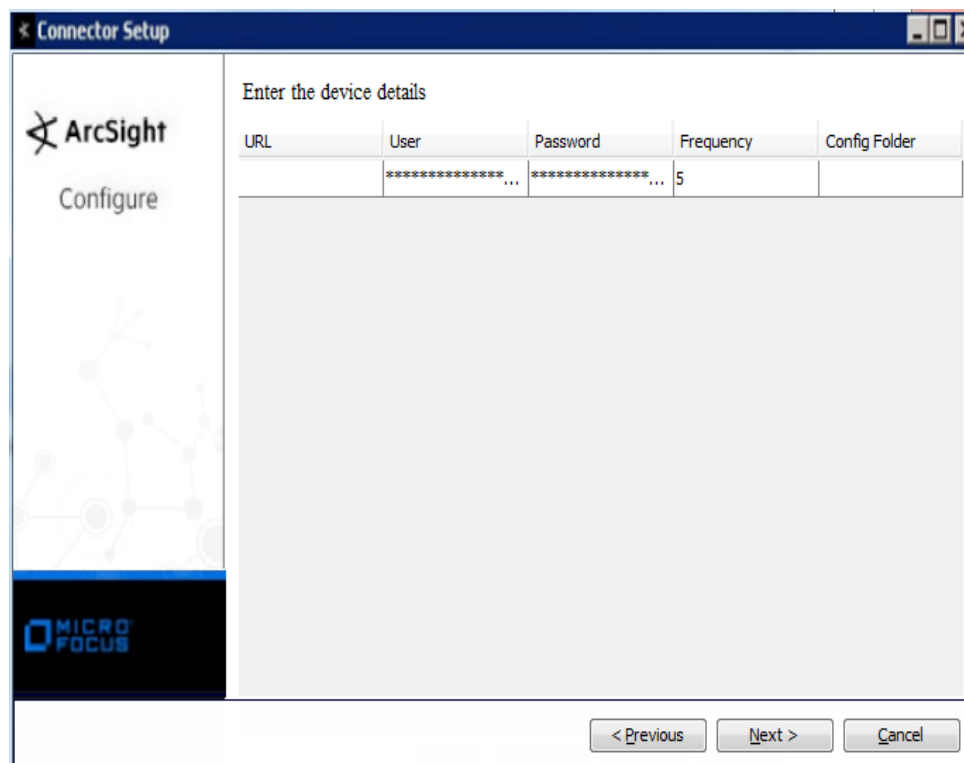
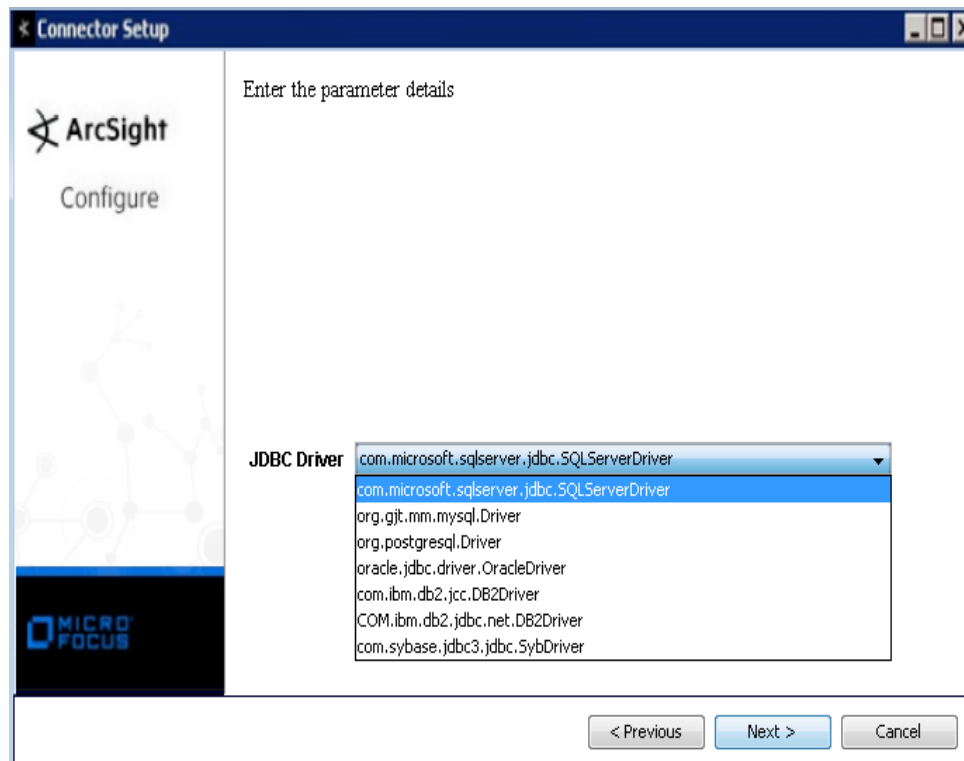
Note: Click 'Export' to copy the host name data you entered in the table to a CSV file. Click 'Import' to select a CSV file and copy it into the table rather than adding the data manually. See the "SmartConnector User's Guide" for more information.

ArcSight FlexConnector Multiple Database

Choose this type to retrieve information from multiple databases that use the same query or retrieve different set of events using different queries from the same database.

**Note:**

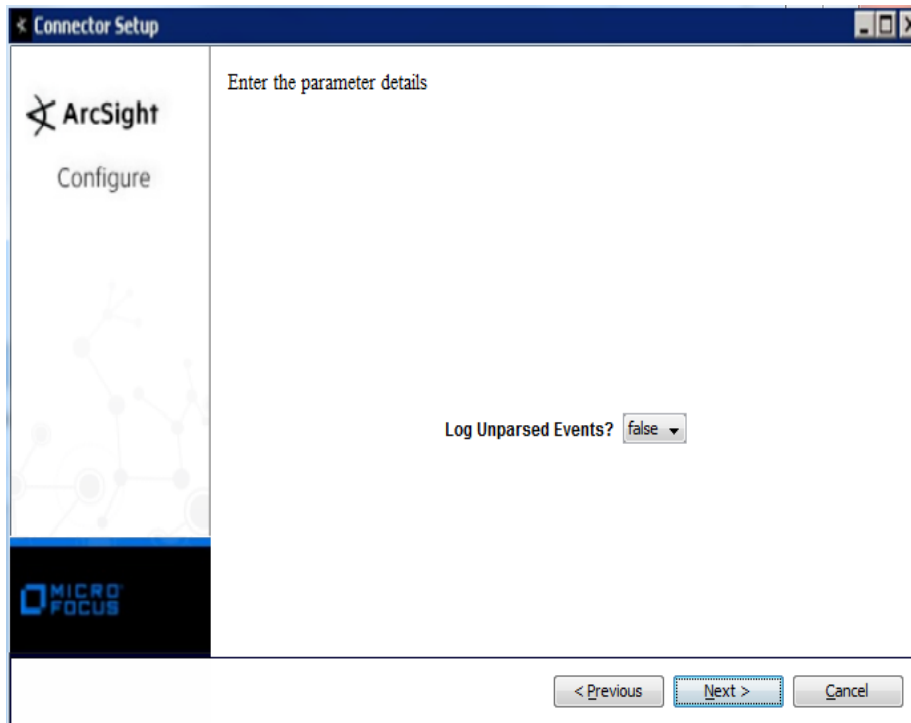
- After installing connector core software and before configuring the ArcSight FlexConnector ID-Based DB, you will need to download an appropriate JDBC driver. See [“Additional Configuration for Database Connectors”](#) for complete information.
- Knowledge of SQL is a prerequisite for coding database FlexConnectors.
- SmartConnector releases since 7.2.1 have implemented Java 8, which does not support ODBC connections; therefore, database connectors can only use JDBC connections. For the same reason, the MS Access database, which uses only ODBC connections, is no longer supported.



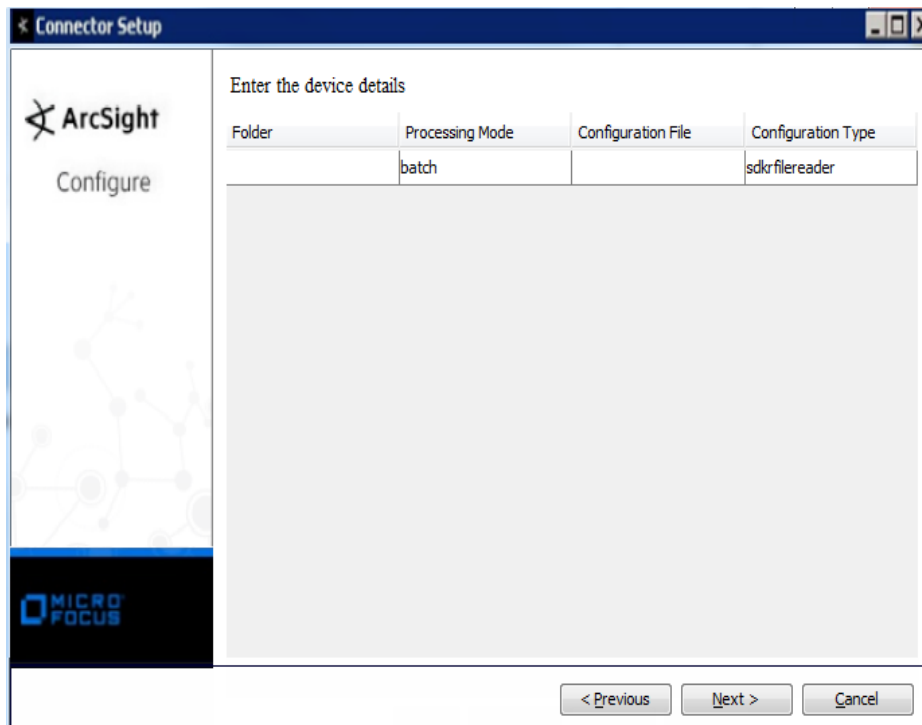
Parameter	Description
JDBC/ODBC Driver	<p>The JDBC driver that will be used to connect to the database.</p> <ul style="list-style-type: none"> For SQL Server, perform the procedure in "Install SQL Server JDBC Driver". For MySQL, perform the procedure in "Install MySQL Driver". For Oracle, use: oracle.jdbc.driver.OracleDriver This default Oracle JDBC driver works with Oracle 9i, 10g, 11g, and 12c database versions. If you are using Oracle 8i, see "Oracle 8i Support". For PostgreSQL, use: org.postgresql.Driver For DB2 unified driver, use: com.ibm.db2.jcc.DB2Driver For DB2 Legacy CLI-based, use: COM.ibm.db2.jdbc.net.DB2Driver For Sybase, use: com.sybase.jdbc2.jdbc.SybDriver
URL	<p>The JDBC URL that identifies the database.</p> <ul style="list-style-type: none"> For Oracle, use: jdbc:oracle:thin:@hostname_or_IP:1521:database_name For MySQL, use: jdbc:mysql://hostname_or_IP:3306/database_name For Microsoft SQL Server 2000, use: jdbc:microsoft:sqlserver://host:port;databasename=name For Microsoft SQL Server 2005 and later, use: jdbc:sqlserver://host:port;databasename=name For PostgreSQL, use: jdbc:postgresql://host/database For DB2 unified driver, use: jdbc:db2:database_name For DB2 Legacy CLI-based, use: jdbc:db2://host_name: port_number/ database_name For Sybase, use: jdbc:sybase:Tds: hostname_or_IP:5000/sybsecurity
User	The database user name.
Password	Password for the database user.
Frequency	Specifies how often, in seconds, to query the database. The default is 5 seconds.
Config Folder	<p>Enter the name of the folder that contains the properties file. Do not enter the full path to the file as doing so will result in an error.</p> <p>This is also the root name of the configuration file. If the configuration folder is myfolder, then the FlexConnector will look for the configuration file in the directory:</p> <p>ARCSIGHT_HOME\user\agent\flexagent\myfolder</p> <ul style="list-style-type: none"> The configuration file for time-based connectors will be named: myfolder.sdktbdatabase.properties The configuration file for ID-based connectors will be named: myfolder.sdkibdatabase.properties

ArcSight FlexConnector Multiple Folder File

This type parses files (fixed, delimited, or using regular expressions) that are written to multiple folders. Events can be read in real time or in batch mode.



The screenshot shows the 'Connector Setup' window for ArcSight. The left sidebar contains the ArcSight logo and a 'Configure' button. The main area is titled 'Enter the parameter details'. It features a 'Log Unparsed Events?' dropdown menu currently set to 'false'. At the bottom, there are three buttons: '< Previous', 'Next >', and 'Cancel'.



The screenshot shows the 'Connector Setup' window for ArcSight, specifically the 'Enter the device details' step. The left sidebar is identical to the previous screenshot. The main area contains a table with the following data:

Folder	Processing Mode	Configuration File	Configuration Type
	batch		sdkrfilereader

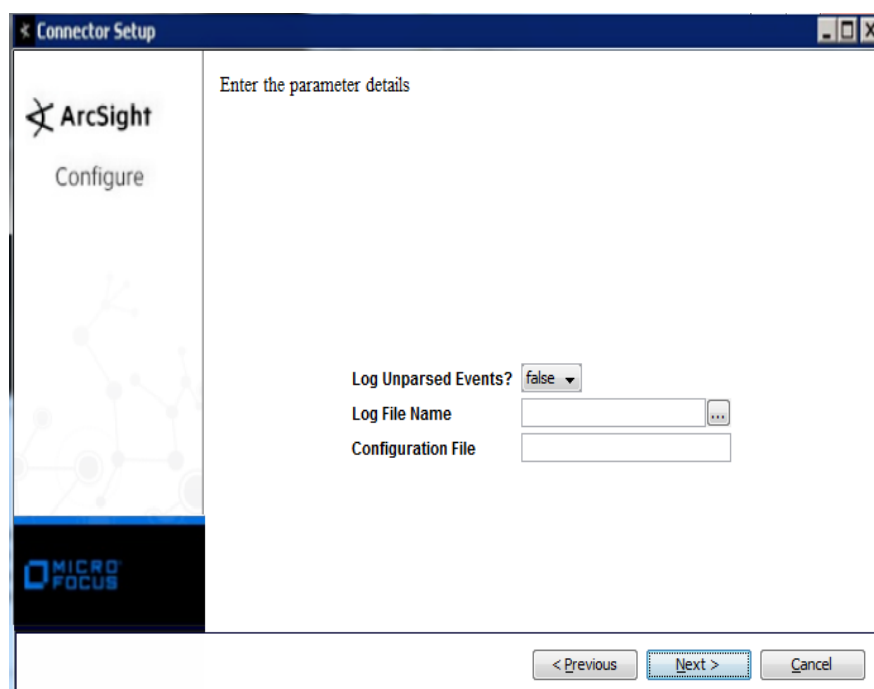
Below the table is a large, empty gray rectangular area. At the bottom, there are three buttons: '< Previous', 'Next >', and 'Cancel'.

Parameter	Description
Log Unparsed Events?	The default value is false. Select true for the connector to detect and log unparsed events to \$ARCSIGHT_HOME\current\logs\events.log. For more information on unparsed events, see “Unparsed Events Detection” .
Folder	The absolute path of the directory where log files for the FlexConnector are located. For example: c:\logs
Processing Mode	If the files in the folder are not being written to in real time and are complete, select batch . If the files are open and new log lines are being added to them, select realtime .
Configuration File	<p>The base name of the configuration file that describes the format of the log file.</p> <ul style="list-style-type: none">For a connector that parses fixed-format files, the suffix .sdkfilereader.properties is appended automatically.For a connector that parses variable-format files, the suffix .sdkrfilereader.properties is appended automatically. <p>For example, if you specify the following name for a configuration file that parses fixed-format log files: sample</p> <p>The filename becomes:</p> <p>ARCSIGHT_HOME\user\agent\flexagent\sample.sdkfilereader.properties</p>
Configuration Type	<ul style="list-style-type: none">If the file is a fixed-format log file, select sdkfilereader.If the file is a variable-format log file, select sdkrfilereader.If the file is a keyvalue-format log file, select sdkkeyvalue.If the file is a CEF-format log file, select cef.

ArcSight FlexConnector Regex File

This type reads variable-format log files. Choose this type if the source log files have one event per line, but the format of each line varies based on the type of event information. In this case, each line shares a common section (for example, the date and hostname), but the number and content of the other fields on the line varies. For devices that may not write to log files in real time, use the ["ArcSight FlexConnector Regex Folder File"](#).

Note: The regular expression-based FlexConnectors require a familiarity with Java-compatible regular expressions.



Parameter	Description
Log Unparsed Events	The default value is false. Select true for the connector to detect and log unparsed events to \$ARCSIGHT_HOME\current\logs\events.log . For more information on unparsed events, see "Unparsed Events Detection" .
Log File Name	The absolute path and name of the file that this FlexConnector will read. For example: c:\temp\sample_data.txt
Configuration File	<p>The base name of the configuration file that describes the format of the log file.</p> <ul style="list-style-type: none"> For a connector that parses fixed-format files, the suffix .sdkfilereader.properties is appended automatically. For a connector that parses variable-format files, the suffix .sdkrfilereader.properties is appended automatically. <p>For example, if you specify the following name for a configuration file that parses fixed-format log files: sample</p> <p>The filename becomes:</p> <p>ARCSIGHT_HOME\user\agent\flexagent\sample.sdkfilereader.properties</p>

ArcSight FlexConnector Regex Folder File

Choose this type to parse log files using regular expressions to which data is not written in real time. This type recursively reads variable-format log files in a folder or multiple folders.

Note: The regular expression-based FlexConnectors require a familiarity with Java-compatible regular expressions.

Parameter	Description
Log Unparsed Events?	The default value is false . Select true for the connector to detect and log unparsed events to \$ARCSIGHT_HOME\current\logs\events.log . For more information on unparsed events, see " Unparsed Events Detection ".
Log Folder	The absolute path of the directory where log files for the FlexConnector are located. For example: c:\logs
Configuration File	<p>The base name of the configuration file describing the format of the log file.</p> <ul style="list-style-type: none"> For a connector that parses fixed-format files, the suffix .sdkfilereader.properties is appended automatically. For a connector that parses variable-format files, the suffix .sdrfilereader.properties is appended automatically. <p>For example, if you specify the following name for a configuration file that parses fixed-format log files: sample</p> <p>The filename becomes:</p> <p>ARCSIGHT_HOME\user\agent\flexagent\sample.sdkfilereader.properties</p>

ArcSight FlexConnector REST

This type uses REST API endpoints, JSON parser, and OAuth2 authentication to collect security events from cloud vendors (such as Salesforce or Google Apps). This FlexConnector is not documented in this guide. See the *ArcSight FlexConnector REST Developer's Guide* for details.

Connector Setup

ArcSight
Configure

Enter the parameter details

Proxy Host

Proxy Port

Proxy User Name

Proxy Password

Configuration File

Events URL

Authentication Type: Basic

User Name

Password

OAuth2 Client Properties File

Refresh Token

< Previous Next > Cancel

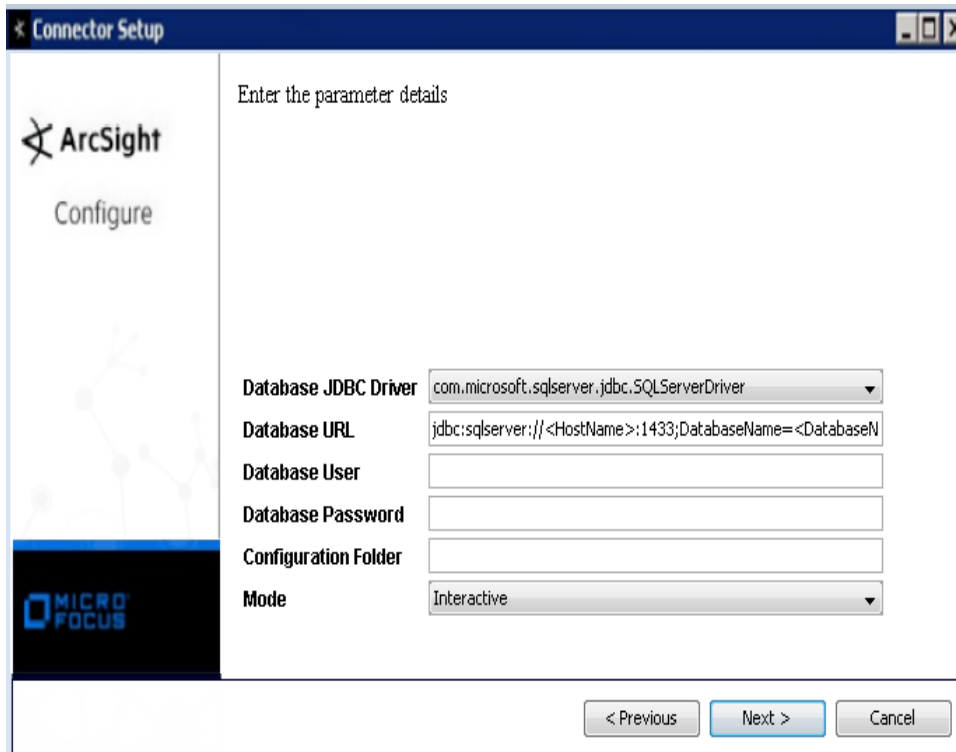
ArcSight FlexConnector Scanner Database

Choose this type to import the results of a scan from a scanner device and forward the data to ESM so that ESM can model an organization's assets, open ports, operating systems, applications, and vulnerabilities. The connector imports periodic scans to ESM, which uses this information for event prioritization, reporting, and correlation.

A database contains results for multiple scans where each scan is identified by a job identifier (ID). The scan results are organized in multiple tables that are linked by job IDs or other IDs. SQL query-based parsers are used to extract relevant information from the scan results.

Note:

- After installing connector core software and before configuring the ArcSight FlexConnector ID-Based Database, you will need to download an appropriate JDBC driver. See ["Additional Configuration for Database Connectors"](#) for complete information.
- Knowledge of SQL is a prerequisite for coding database FlexConnectors.
- SmartConnector releases since 7.2.1 have implemented Java 8, which does not support ODBC connections; therefore, database connectors can only use JDBC connections. For the same reason, the MS Access database, which uses only ODBC connections, is no longer supported.



The screenshot shows the "Connector Setup" window. On the left is a sidebar with the ArcSight logo and a "Configure" button. The main area is titled "Enter the parameter details" and contains several input fields:

- Database JDBC Driver:** A dropdown menu showing "com.microsoft.sqlserver.jdbc.SQLServerDriver".
- Database URL:** A text field containing "jdbc:sqlserver://<HostName>:1433;DatabaseName=<DatabaseN".
- Database User:** An empty text field.
- Database Password:** An empty text field.
- Configuration Folder:** An empty text field.
- Mode:** A dropdown menu showing "Interactive".

At the bottom right are three buttons: "< Previous", "Next >", and "Cancel".

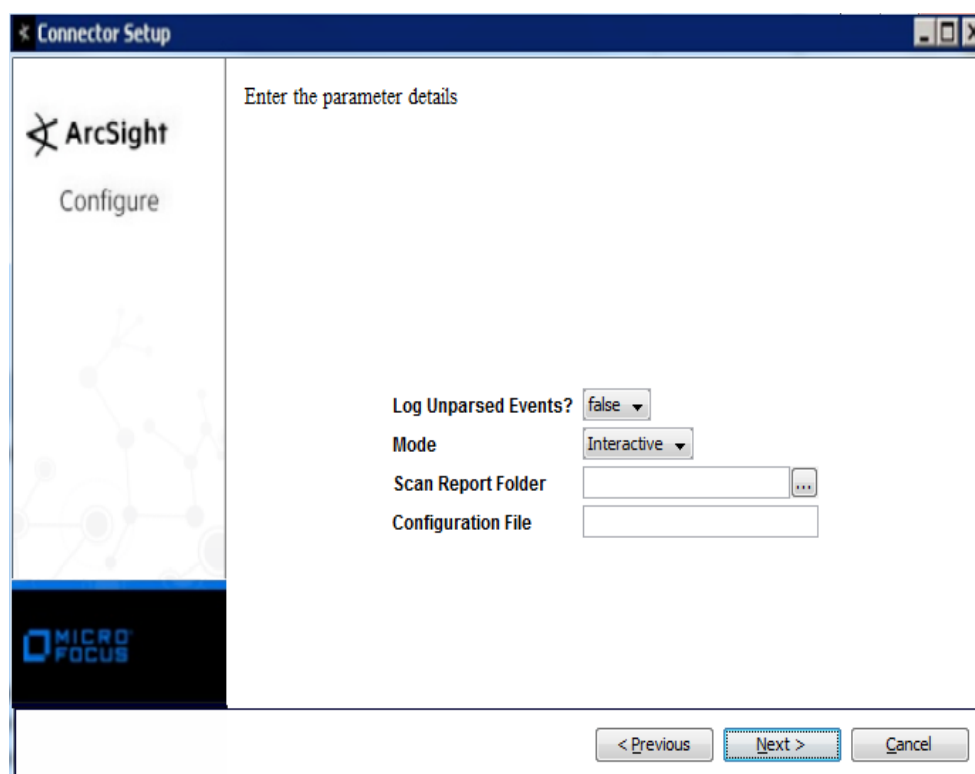
Parameter	Description
Database JDBC Driver	<p>The JDBC driver that will be used to connect to the database.</p> <ul style="list-style-type: none"> For SQL Server, perform the procedure in "Install SQL Server JDBC Driver". For MySQL, perform the procedure in "Install MySQL Driver". <p>For Oracle, use: oracle.jdbc.driver.OracleDriver</p> <p>This default Oracle JDBC driver works with Oracle 9i, 10g, 11g, and 12c database versions. If you are using Oracle 8i, see "Oracle 8i Support".</p> <ul style="list-style-type: none"> For PostgreSQL, use: org.postgresql.Driver For DB2 unified driver, use: com.ibm.db2.jcc.DB2Driver For DB2 Legacy CLI-based, use: COM.ibm.db2.jdbc.net.DB2Driver For Sybase, use: com.sybase.jdbc2.jdbc.SybDriver
Database URL	<p>The JDBC URL that identifies the database.</p> <ul style="list-style-type: none"> For Oracle, use: jdbc:oracle:thin:@hostname_or_IP:1521:database_name For MySQL, use: jdbc:mysql://hostname_or_IP:3306/database_name For Microsoft SQL Server 2000, use: jdbc:microsoft:sqlserver://host:port;databasename=name For Microsoft SQL Server 2005 and later, use: jdbc:sqlserver://host:port;databasename=name For PostgreSQL, use: jdbc:postgresql://host/database For DB2 unified driver, use: jdbc:db2:database_name For DB2 Legacy CLI-based, use: jdbc:db2://host_name: port_number/ database_name For Sybase, use: jdbc:sybase:Tds: hostname_or_IP:5000/sybsecurity
Database User	The database user name.

Parameter	Description
Database Password	Password for the database user.
Configuration Folder	<p>Enter the name of the folder that contains the properties file. Do not enter the full path to the file as doing so will result in an error.</p> <p>This is also the root name of the configuration file. If the configuration folder is "myfolder", the FlexConnector will look for the configuration file in the directory:</p> <p>ARCSIGHT_HOME\user\agent\flexagent\myfolder</p> <ul style="list-style-type: none">• The configuration file for time-based connectors will be named: myfolder.sdktdatabase.properties• The configuration file for ID-based connectors will be named: myfolder.sdkibdatabase.properties
Mode	<ul style="list-style-type: none">• If the files in the folder are not being written to in real time and are complete, select batch.• If the files are open and new log lines are being added to them, select realtime.

ArcSight FlexConnector Scanner Text Reports

Choose this type to import the results of a scan from a scanner device and forward the data to ESM so that ESM can model an organization's assets, open ports, operating systems, applications, and vulnerabilities. The connector imports periodic scans to ESM, which uses this information for event prioritization, reporting, and correlation.

A normal text report contains results for a single scan with each line in the report containing a piece of information about a host. Regular expression based parsers are used to extract relevant information from the report



Parameter	Description
Log Unparsed Events	The default value is false. Select true for the connector to detect and log unparsed events to <code>\$ARCSIGHT_HOME\current\logs\events.log</code> . For more information on unparsed events, see "Unparsed Events Detection" .
Mode	<ul style="list-style-type: none"> If the files in the folder are not being written to in real time and are complete, select batch. If the files are open and new log lines are being added to them, select realtime.
Scan Report Folder	The folder in which the scanner reports are located.
Configuration File	<p>The base name of the configuration file describing the format of the log file.</p> <ul style="list-style-type: none"> For a connector that parses fixed-format files, the suffix <code>.sdkfilereader.properties</code> is appended automatically. For a connector that parses variable-format files, the suffix <code>.sdkrfilereader.properties</code> is appended automatically. <p>For example, if you specify the following name for a configuration file that parses fixed-format log files: sample</p> <p>The filename becomes:</p> <p>ARCSIGHT_HOME\user\agent\flexagent\sample.sdkfilereader.properties</p>

ArcSight FlexConnector Scanner XML Reports

Choose this type to import the results of a scan from a scanner device and forward the data to ESM so that ESM can model an organization's assets, open ports, operating systems, applications, and vulnerabilities.

The connector imports periodic scans to ESM, which uses this information for event prioritization, reporting, and correlation.

An XML report contains results for a single scan with scan results organized in the form of nested XML elements. XQuery/XPath-based parsers are used to extract relevant information from the report.

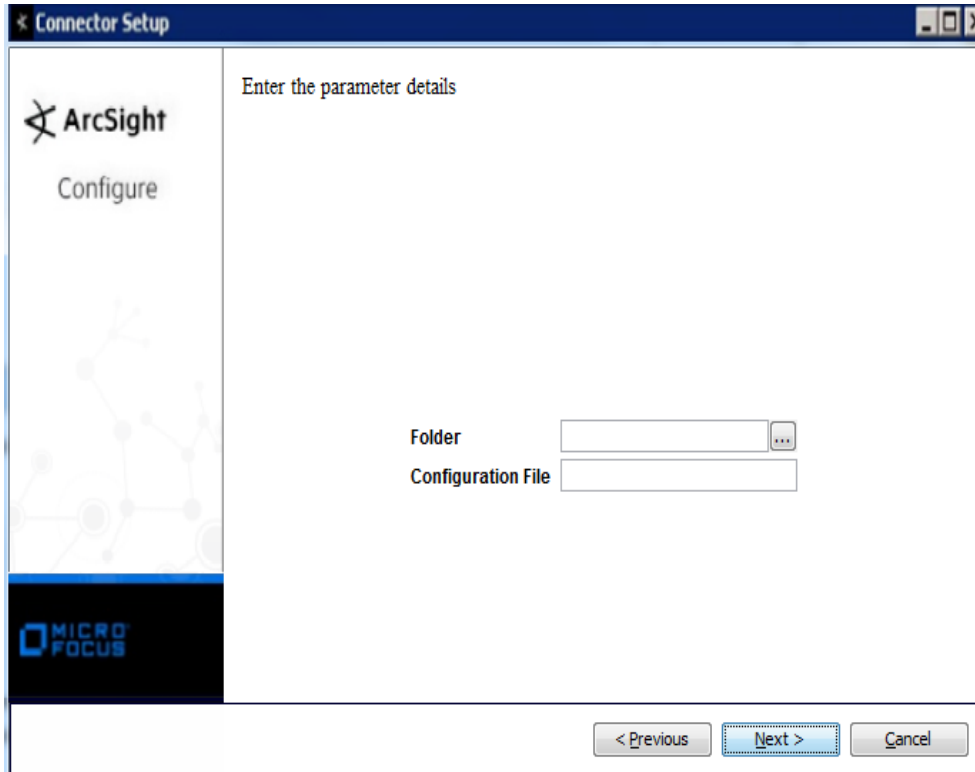
Note: The XML FlexConnectors require a familiarity with XML, XPath, and XQuery.

Parameter	Description
Mode	<ul style="list-style-type: none"> If the files in the folder are not being written to in real time and are complete, select batch. If the files are open and new log lines are being added to them, select realtime.
Report Folder	The folder in which the SAINT scanner reports are located.
Configuration File	<p>The base name of the configuration file describing the format of the log file.</p> <ul style="list-style-type: none"> For a connector that parses fixed-format files, the suffix .sdkfilereader.properties is appended automatically. For a connector that parses variable-format files, the suffix .sdkrfilereader.properties is appended automatically. <p>For example, if you specify the following name for a configuration file that parses fixed-format log files: sample</p> <p>The filename becomes:</p> <p>ARCSIGHT_HOME\user\agent\flexagent\sample.sdkfilereader.properties</p>

ArcSight FlexConnector XML File

Choose this type for devices that write event information to XML files. Event information in these files is presented in standard XML format, using namespaces, elements, attributes, text, and cdata. The connector recursively reads the events from the XML-based files in a folder.

Note: The XML FlexConnectors require a familiarity with XML, XPath, and XQuery.



Parameter	Description
Folder	The absolute path of the directory where log files for the FlexConnector are located. For example: c:\logs
Configuration File	<p>The base name of the configuration file describing the format of the log file.</p> <ul style="list-style-type: none">For a connector that parses fixed-format files, the suffix .sdkfilereader.properties is appended automatically.For a connector that parses variable-format files, the suffix .sdkrfilereader.properties is appended automatically. <p>For example, if you specify the following name for a configuration file that parses fixed-format log files: sample</p> <p>The filename becomes: ARCSIGHT_HOME\user\agent\flexagent\sample.sdkfilereader.properties</p>

ArcSight FlexConnector Simple Network Management Protocol (SNMP Unified)

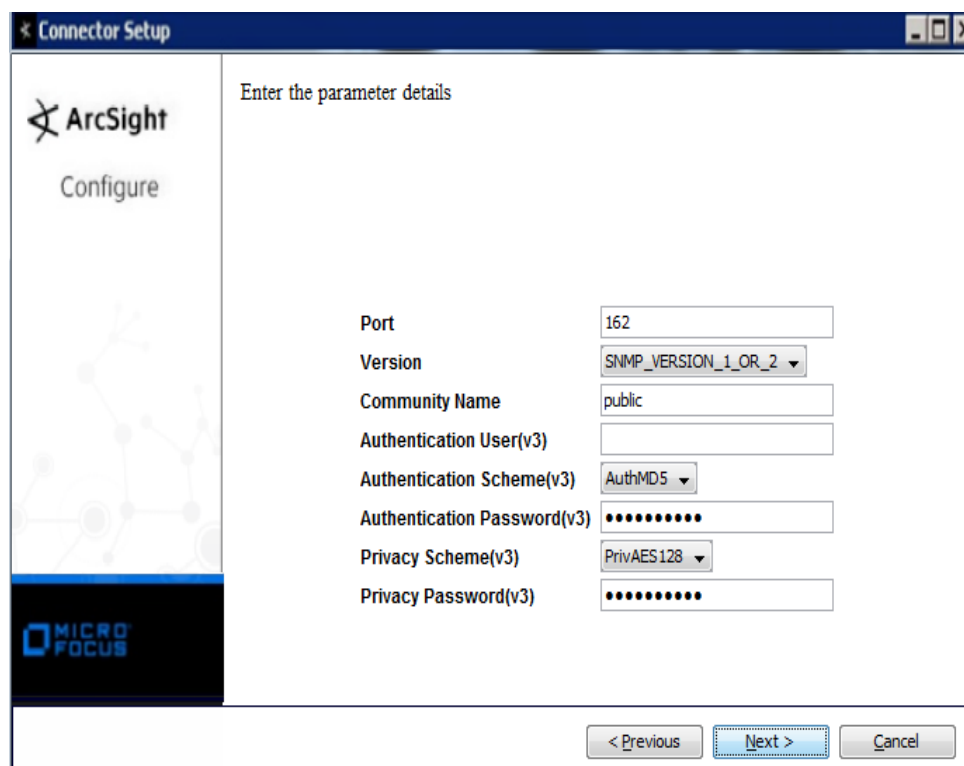
To install a connector to collect event information from SNMP traps, select **SNMP Unified** from the list of SmartConnector to install. The SmartConnector for SNMP Unified requires that you create a specific folder and copy your parser into that folder. After installation of core connector software, under **\$ARCSIGHT_HOME/current/user/agent**, create the following subfolder if it does not already exist. This folder is for the various trap OIDs.

flexagent/snmp/subfolder

For example:

\$ARCSIGHT_HOME/current/user/agent/flexagent/snmp/<trap OID>

For complete connector installation and configuration information, see the configuration guide for the *SmartConnector for SNMP Unified*.



The screenshot shows the 'Connector Setup' window for ArcSight. The title bar reads '< Connector Setup'. On the left is a sidebar with the ArcSight logo and the word 'Configure'. The main area is titled 'Enter the parameter details'. It contains the following fields:

Parameter	Value
Port	162
Version	SNMP_VERSION_1_OR_2
Community Name	public
Authentication User(v3)	
Authentication Scheme(v3)	AuthMD5
Authentication Password(v3)
Privacy Scheme(v3)	PrivAES128
Privacy Password(v3)

At the bottom are three buttons: '< Previous', 'Next >', and 'Cancel'.

ArcSight FlexConnector Syslog

Select the **Syslog Daemon** connector from the list of SmartConnector to install if you want to create a Syslog FlexConnector.

Parameter	Description
Network Port	The port the connector listens to for syslog events.
IP	The connector listens for syslog events only from this IP address. Enter (ALL) for all IP addresses in the specified port address.
Protocol	Select UDP or Raw TCP as the protocol to be used to receive events.
Forwarder	<p>The CEF Forwarder mode parameter is false by default. If the destination is a Syslog Daemon connector and you want to preserve information about the original connector, then the CEF Forwarder mode should be set to true both in this destination and in the receiving connector. That is, if you have a chain of connectors connected by syslog, syslog NG, or CEF encrypted syslog (UDP), and you want to preserve information about the original connector, the destinations should all have the CEF Forwarder mode set to true (which is implicitly true for CEF Encrypted Syslog (UDP)), and the connectors receiving from them should also have the CEF Forwarder mode set to true.</p> <p>For example, you can configure a number of connectors to all send events using the CEF Syslog destination type to one Syslog Daemon connector, which then sends to ESM. In order for the events arriving at ESM to retain information about the specific connector that collected the event, the connector's CEF Syslog destinations should have the Forwarder mode set to true, and the Syslog Daemon connector should also set the Forwarder mode to true. The information will display in the original agent fields of the events.</p>

Select a Destination

This section describes selecting the ArcSight Manager (encrypted) destination. For information about this destination or any of the other possible destinations, see the *ArcSight SmartConnector User Guide*.

1. The next window asks for the destination type; make sure **ArcSight Manager (encrypted)** is selected and click **Next**.
2. Enter values for the **Manager Host Name**, **Manager Port**, **User**, and **Password** required parameters. This is the same ArcSight user name and password you created during the ArcSight Manager installation. Click **Next**.
3. Enter a name for the SmartConnector and provide other information identifying the connector's use in your environment. Click **Next**. The connector starts the registration process.
4. The certificate import window for the ArcSight Manager is displayed. Select **Import the certificate to the connector from destination** and click **Next**. (If you select **Do not import the certificate to connector from destination**, the connector installation will end.) The certificate is imported and the **Add connector Summary** window is displayed.

Complete Installation and Configuration

1. Review the **Add Connector Summary** and click **Next**. If the summary is incorrect, click **Previous** to make changes.
2. The wizard now prompts you to choose whether you want to run the SmartConnector as a stand-alone process or as a service. If you choose to run the connector as a stand-alone process, select **Leave as a standalone application**, click **Next**, and continue with step 5.
3. If you chose to run the connector as a service, with **Install as a service** selected, click **Next**. The wizard prompts you to define service parameters. Enter values for **Service Internal Name** and **Service Display Name** and select **Yes** or **No** for **Start the service automatically**. The **Install Service Summary** window is displayed when you click **Next**.
4. Click **Next** on the summary window.
5. To complete the installation, choose **Exit** and click **Next**.

Additional Configuration for Database Connectors

For database connectors, connection to the database through a JDBC driver is required. The following sections provide instructions for downloading and installing JDBC drivers, instructions for adding JDBC drivers to Connector Appliance/ArcSight Management Center, and how to use a JDBC driver with connectors that connect to Microsoft SQL Servers using Windows Authentication only:

- [Install SQL Server JDBC Driver](#)
- [Install MySQL Driver](#)
- [Add a JDBC Driver to the Connector Appliance/ArcSight Management Center](#)
- [Configure the JDBC Driver and Windows Authentication](#)

Some changes are required if you are using an Oracle 8i database. See:

- [Oracle 8i Support](#)

To avoid duplicate events when developing a new connector, see:

- [Troubleshooting Duplicate Events](#)

See "Why does my connection to SQL Server fail/hang?" in "[Frequently Asked Questions](#)" for information on possible database connection issues that can result from certain Java versions.

Install SQL Server JDBC Driver

There are a number of steps that must be completed outside of FlexConnector setup before the connector will be able to establish a connection with an SQL Server.

First, a Microsoft SQL Server JDBC Driver must be downloaded. For information about and to download the MS SQL Server JDBC Driver, see:

<http://msdn.microsoft.com/en-us/sqlserver/aa937724>

Different versions of the JDBC driver are required for different SQL Server database versions; be sure to use the correct driver for your database version. The name of the jar file may be different for some JDBC driver versions.

When you download the JDBC driver, the version of the jar file depends on the version of the JRE the connector uses:

Version 7.2.1 and later use JRE 1.8 and require `sqljdbc42.jar` (available with JDBC Driver 6.0 for SQL Server)

Version 7.1.2 and later use JRE 1.7 and require `sqljdbc41.jar` (available with JDBC Driver 6.0 for SQL Server)

Prior versions, which run JRE 1.6, require `sqljdbc4.jar` (available with JDBC Driver 4.0 for SQL Server)

Install the JDBC driver:

After installation of SmartConnector core software, when the "Add a Connector" window is displayed, perform the following tasks:

1. Copy the **sqljdbc** jar file to **\$ARCSIGHT_HOME\current\user\agent\lib**. The version of the jar file depends on the version of the Java Runtime Environment (JRE) the connector uses. SmartConnector versions 7.1.2 and later use JRE 1.7 (also referred to as Java 7) and require **sqljdbc41.jar**. Prior versions of connectors that run JRE 1.6 (also referred to as Java 6) require **sqljdbc4.jar**.
2. If using Windows authentication copy **sqljdbc_auth.dll** to **\$ARCSIGHT_HOME\current\jre\bin**. Note that there are additional prerequisites that are outside the scope of ArcSight for Windows authentication to work, and confirming connectivity with an SQL Server user might prove easier during the initial setup (Microsoft recommends Windows Authentication).
3. Copy the FlexConnector Configuration File (such as **.sdkibdatabase.properties**) to **\$ARCSIGHT_HOME\user\agent\flexagent\<Configuration Folder>** (the FlexConnector

database parameter configuration folder). The setup will run queries in the FlexConnector configuration file as part of verification of the connector parameters. One of the queries is **maxid.query** for id-based connectors and **lastdate.query** for time-based connectors, which will set **MaxID** or **lastdate.query** for the FlexConnector; when the FlexConnector is started, it will only process rows added after **MaxID** or **lastdate.query**.

4. From the **\$ARCSIGHT_HOME/current/bin** directory, double-click **runagentsetup** to return to the SmartConnector Configuration Wizard. At this point FlexConnector setup can be completed. Below are some additional things to consider when configuring the following connector parameters.

- The database JDBC Driver must be

com.microsoft.sqlserver.jdbc.SQLServerDriver

- The database URL should be similar to

jdbc:sqlserver://host:port;databasename=db (append **;integratedSecurity=true** if using Windows authentication).

For Microsoft SQL Server 2000, use:

jdbc:microsoft:sqlserver://host:port;databasename=name

For Microsoft SQL Server 2005 and later, use:

jdbc:sqlserver://host:port;databasename=name

- The database user must include the domain if using Windows authentication and the connector host and the SQL Server host are in different domains.
- The database Password must correspond to the database user (SQL Server or Windows).
- The configuration folder must correspond to the folder where the FlexConnector configuration file is placed (see step 3 above).

When using the ArcSight Connector Appliance or ArcSight Management Center, after downloading and extracting the JDBC driver, upload the driver into the repository and apply it to the appropriate container or containers. See "[Add a JDBC Driver to the Connector Appliance/ArcSight Management Center](#)" for detailed information.

Install MySQL Driver

After installing connector core software and before configuring the ArcSight FlexConnector ID-based DB, you will need to follow these instructions to download an appropriate JDBC driver.

1. Click **Cancel** to leave the configuration wizard at this point.
2. The following steps are required when you use the MySQL JDBC driver, required for Connector Appliance/ArcSight Management Center and Linux systems.
 - a. For connector versions 7.2.4 and later, download the latest MySQL JDBC Driver from:

<http://dev.mysql.com/downloads/connector/j>

For connector versions 7.2.3 and earlier, download the MySQL 5.0.8 JDBC Driver from:

<https://dev.mysql.com/downloads/connector/j/5.0.html>

Install the driver.

- b. For software connectors, copy the appropriate jar file to **\$ARCSIGHT_HOME\current\user\agent\lib**, where **\$ARCSIGHT_HOME** refers to the connector install folder, such as **c:\ArcSight\SmartConnectors**. For Connector Appliance/ArcSight Management Center users, see "[Add a JDBC Driver to the Connector Appliance/ArcSight Management Center](#)".
- c. From **\$ARCSIGHT_HOME/current/bin**, double-click **runagentsetup** to return to the SmartConnector Configuration Wizard.

Add a JDBC Driver to the Connector Appliance/ArcSight Management Center

1. After downloading and extracting the JDBC driver, upload the driver into the repository and apply it to the appropriate container or containers, as described in this section.
2. From the Connector Appliance/ArcSight Management Center, select **Setup > Repositories**.
3. Select **JDBC Drivers** from the left pane and click the **JDBC Drivers** tab.
4. Click **Upload to Repository**.
5. From the Repository File Creation Wizard, select **Individual Files**, then click **Next**.
6. Retain the default selection and click **Next**.
7. Click **Upload** and locate and select the **.jar** file you downloaded in step 3 of SmartConnector Installation.
8. Click **Submit** to add the specified file to the repository and click **Next** to continue.
9. After adding all files you require, click **Next**.
10. In the **Name** field, enter a descriptive name for the zip file (JDBCdriver, for example). Click **Next**.
11. Click **Done** to complete the process; the newly added file is displayed in the **Name** field under **Add Connector JDBC Driver File**.
12. To apply the driver file, select the driver **.zip** file and click the up arrow to invoke the Upload Container Files wizard. Click **Next**.
13. Select the container or containers into which the driver is to be uploaded; click **Next**.
14. Click **Done** to complete the process.
15. Add the connector through the Connector Appliance/ArcSight Management Center interface; see the Connector Appliance/ArcSight Management Center Online Help for detailed information. Descriptions of parameters to be entered during connector configuration are provided in "[Select Connector and Add Parameter Information](#)".

Configure the JDBC Driver and Windows Authentication

This section provides guidance on how to use a JDBC driver with SmartConnectors that connect to Microsoft SQL Servers using Windows Authentication only. As previously described, download the SQL JDBC drivers from Microsoft and install the driver before beginning this procedure.

Note:

- The JDBC driver does not provide function to supply Windows authentication credentials such as user name and password. In such cases, the applications must use SQL Server Authentication. When installing the connector on a non-Windows platform, configure the Microsoft SQL Server for Mixed Mode Authentication or SQL Server Authentication.
- Microsoft Type 4 JDBC drivers (versions 4.0 or later) support integrated authentication. Windows Authentication works only when using one of these drivers. You also must add **;integratedSecurity=true** to the JDBC URL entry for the connection to your database.

1. Copy the **sqljdbc_auth.dll** file from the JDBC driver download to the **\$ARCSIGHT_HOME\jre\bin** directory. For example, the JDBC driver download path for SQL JDBC driver version 4.0 for 32-bit environment would be **sqljdbc_4.0\enu\auth\x86\sqljdbc_auth.dll** and, for 64-bit environment, **sqljdbc_4.0\enu\auth\x64\sqljdbc_auth.dll**.

Note: When upgrading a connector, the **\$ARCSIGHT_HOME\jre\bin** directory is overwritten, therefore, you must copy the **sqljdbc_auth.dll** authentication file to this folder again after update.

2. Go to **\$ARCSIGHT_HOME\current\bin** and double-click **runagentsetup** to continue the SmartConnector installation.
3. When entering the connector parameters, in the **JDBC Database URL** field, append **;integratedSecurity=true** to the end of the URL string. In the following example, note that the name or instance of the database configured at installation/audit time should be used.

```
jdbc:sqlserver://mysqlserver:1433;DatabaseName=mydatabase;integratedSecurity=true
```

4. Complete the remaining connector wizard configuration steps.
5. After completing the connector installation, if running on a Windows Server, change the service account to use the Windows account that should login to the database. The connector will use the account used to start the service, regardless of the account value setting entered in the connector setup process.

Oracle 8i Support

With the addition of Oracle 11g support, ArcSight replaced the 10.2.0.1 **oracle-jdbc** driver in **\$ARCSIGHT_HOME\current\lib\agent** with the **oracle-jdbc-11.1.0.6.jar**. This driver no longer connects to Oracle 8i databases; therefore, before upgrading the connector:

1. Go to **\$ARCSIGHT_HOME\Current\lib\agent** and locate the **oracle-jdbc-10.2.0.1.jar** file. Copy it to a temporary location.
2. After completing connector upgrade and before running the connector, replace the **11.1.0.6.jar** file with the **10.2.0.1.jar** file.

Troubleshooting Duplicate Events

This section provides guidelines that can be used to troubleshoot duplicate events or to avoid duplicate events when developing a new connector

Duplicate events are ignored and not forwarded to ESM or other destinations. Duplicate events caused by the connector can result in lost events. Reasons for connector-caused duplicate events include: primary key not used as ID field, **uniqueid.fields** that are not unique to only one event, and incorrect queries.

Typical parser queries can be divided into two groups:

- simple main query - queries one event table or view.
- complex main query - queries one event table or view with left outer join to secondary tables, views and sub-queries.

Some duplicate events can originate in the connector's parser with either of the following:

- Main query
- **Id.field** and **unique.idfields** for the ID-based DB connector or the **timestamp.field** and **unique.idfields** for the time-based DB connector

If the combination of fields is not unique for each event, then duplicate events will occur.

A **uniqueid.field** can be one or more table fields separated by commas.

You can identify duplicate events by errors in the agent log file such as the following:

```
[..][ERROR][...][processQuery] Event with duplicate ID ..., ignoring
```

Example 1: ID-based Database Connectors Only

This example is for ID-based database connectors only and shows a simple main query with **id** field.

```
query= select evt. ID, evt.SourceHost,... FROM Events as evt
WHERE evt.ID > ? order by ID
id.field=ID
```

Usually, the ID used in the where clause condition and the `id.field` should be the table's primary key.

If a duplicate event occurs, that means the **id.field** is not the primary key. To fix the issue:

- If possible, change the **id.field** to be the primary key.
- If the **id.field** cannot be changed to become a unique primary key for each event, add one or more table fields to the **uniqueid.field** so that the **id.field** and **uniqueid.field** combination is unique for each event.

Example 2: ID-based and Time-based Connectors

For ID-based database connectors:

```
query= select evt. ID, evt.IDX,... FROM Events as evt
WHERE evt.ID > ? order by ID
id.field=ID
uniqueid.field=IDX
```

If duplicate events occur, then the **id.field** is not the primary key and the combination of **id.field** and **unique.idfield** is also not unique to each event. To fix the issue, you should extend **uniqueid.field** to add more fields to it. Add one more field to **uniqueid.field** and then test the connector until the **Event with duplicate ID** error messages do not occur.

For time-based database connectors:

```
query=select evt.ReceivedTime, evt.IDX,... FROM Events as evt
WHERE evt.ReceivedTime >= ? order by evt.ReceivedTime
timestamp.field= ReceivedTime
uniqueid.field=IDX
```

If duplicate events occur, then the **timestamp.field** is not the primary key and the combination of **timestamp.field** and **unique.idfield** is also not unique to each event. To fix the issue, you should extend **uniqueid.field** to add more fields to it. Add one more field to **uniqueid.field** and then test the connector until the **Event with duplicate ID** error messages do not occur.

Example 3: Complex Main Query with a Join

This example is for a complex main query with a join.

```
select evt. ID, etype. EventTypeID ,etype.EventName FROM Events as evt
Left Join EventType as etype on evt.EventTypeID=etype.EventTypeID
WHERE evt.AutoID > 0 order by ID
```

The following tables shows the **join** condition relationship between **evt.EventTypeID** and **etype.EventTypeID**.

If **evt.EventTypeID** is a "many-to-one" or "one-to-one" relationship with **etype.EventTypeID** as shown in the following table:

evt.EventTypeID	etype.EventTypeID	etype.EventName
1	1	select
1	2	update
2		

The query result will be the same number of events as in the Events table and no duplicate events as shown in the following table.

evt.EventTypeID	etype.EventTypeID	etype.EventName
1	1	select
1	1	select
2	2	update

However, if **evt.EventTypeID** is “one-to-many” relationship to **etype.EventTypeID** as shown in the following table:

evt.EventTypeID	etype.EventTypeID	etype.EventName
1	1	select
2	1	insert
	2	update

The query result will be one more event as compared to the Events table and a duplicate event will happen as shown in the following table:

evt.EventTypeID	etype.EventTypeID	etype.EventName
1	1	select
1	1	insert
2	2	update

One way to find out, if the duplicate event is caused by the **join** condition, is to run two queries: the original query and the query without the join:

```
select evt. ID, etype. EventTypeID ,etype.EventName FROM Events as evt
Left Join EventType as etype on evt.EventTypeID=etype.EventTypeID
WHERE evt.AutoID > 0 order by ID
```

Without the join:

```
select evt. ID FROM Events as evt WHERE evt.AutoID > 0 order by ID
```

If the total number of rows returned by the original query is equal to the query without the join, then the duplicate event is not caused by the **join** condition. You can then debug the duplicate event error using [Example 1: ID-based Database Connectors Only](#) and [Example 2: ID-based and Time-based Connectors](#).

If the total number of rows returned by the original query is greater than the query without the join, then the issue is caused by **join** condition and the query must be modified to fix the duplicate event.

Chapter 4: Create a Configuration File

The configuration file (also referred to as a parser) is a text file containing properties (name, value pairs) that describe how the FlexConnector parses event data. Blank lines and lines beginning with the comment character '#' are ignored. Other lines consist of a name, an equal sign, and a value.

Note: Parsers are obfuscated for security reasons. Contact Customer Support for assistance with parser overrides.

The REST FlexConnector is documented in the ArcSight *REST FlexConnector Developer's Guide*.

This chapter contains the following information:

- [Parser File Locations and Names](#)
- [Example Parser File](#)
- [Parser File Structure](#)
- [FlexConnector Creation Wizard for Delimited Log Files](#)
- [Regex Tool for Regex FlexConnectors](#)
- [Start the FlexConnector](#)

Parser File Locations and Names

The following table describes the location and filename of the configuration file used for each type of FlexConnector. The vendor or database is usually named for the device vendor (such as "superSecure").

Type	Location	Filename
Log file	ARCSIGHT_HOME\user\agent\ flexagent	vendor.sdkfilereader.properties
Regex Log file	ARCSIGHT_HOME\user\agent\ flexagent	vendor.sdkrfilereader.properties
Regex Folder Follower	ARCSIGHT_HOME\user\agent\ flexagent	vendor.sdkrfilereader.properties
Time-based Database	ARCSIGHT_HOME\user\agent\ flexagent\vendor or product_name	database.sdktbdatabase.properties
ID-based Database	ARCSIGHT_HOME\user\agent\ flexagent\vendor or product_name	database.sdkibdatabase.properties
Multi-Database	ARCSIGHT_HOME\user\agent\ flexagent\vendor or product_name	database.sdktbdatabase.properties

Type	Location	Filename
SNMP	For the SmartConnector for SNMP Unified, create folders for your various trap OIDs as follows: ARCSIGHT_HOME\user\agent\ flexagent\snmp\<various trap OIDs>	sdksnmp.#.snmptrap.properties (# = trap type, such as '1') and sdksnmp.#.sdksnmptrap.properties (varies from the type sdksnmp.#.snmptrap in that trap types must be defined in the parser using the token trap.types)
Syslog	ARCSIGHT_HOME\user\agent\flexagent\syslog	vendor.subagent.sdkfilereader.properties
XML Folder Follower	ARCSIGHT_HOME\user\agent\flexagent	vendor.xqueryparser.properties
JSON Folder Follower	ARCSIGHT_HOME\user\agent\flexagent	vendor.jsonparser.properties
Scanner (for normal text)	ARCSIGHT_HOME\user\agent\flexagent	vendor.scanner.sdkfilereader.properties vendor.vulns.sdkfilereader.properties See also " Getting Vulnerabilities for Scanned Hosts ". vendor.openports.sdkfilereader.properties See also " Getting Open Ports on Scanned Hosts ". vendor.uris.sdkfilereader.properties See also " Getting OS and Applications (URIs) on Scanned Hosts ".
Scanner (for XML)	ARCSIGHT_HOME\user\agent\flexagent	vendor.scanner.xqueryparser.properties vendor.vulns.xqueryparser.properties vendor.openports.xqueryparser.properties vendor.uris.xqueryparser.properties
Scanner (for database)	ARCSIGHT_HOME\user\agent\flexagent\vendor or product_name	database.sdkdatabase.properties

Example Parser File

FlexConnectors are controlled by a configuration file, which is described in more detail the examples shown in "[Configuration File Examples](#)". The following example illustrates a simple Log File FlexConnector configuration file:

```

comments.start.with=#
delimiter=,
token.count=5
token[0].name=Time_of_the_event
token[0].type=TimeStamp
token[0].format=yyyy-MM-dd HH:mm:ss

```

```
token[1].name=ClientIp
token[1].type=IPAddress
token[2].name=Method
token[2].type=String
token[3].name=URL
token[3].type=String
token[4].name=Status
token[4].type=String

event.deviceReceiptTime=Time_of_the_event
event.sourceAddress=ClientIp
event.deviceSeverity=Status
event.requestUrl=URL
event.requestMethod=Method

event.deviceVendor=__getVendor("MyVendor")
event.deviceProduct=__stringConstant("MyProduct")

severity.map.veryhigh.if.deviceSeverity=404,500
severity.map.medium.if.deviceSeverity=303,302
severity.map.low.if.deviceSeverity=200..204
```

Parser File Structure

The type of information a configuration file contains depends on its FlexConnector type. However, the following information types are common to all types of FlexConnectors:

- [Token Declarations](#) (Tokenization)
- [Event Mapping](#) (Normalization)
- [Severity Mapping](#)
- [Extra Processors](#)

Here is an example of a configuration file that contains the most common information types.

```
token.count=5
token[0].name=Time_of_the_event
token[0].type=TimeStamp
token[0].format=yyyy-MM-dd HH:mm:ss
token[1].name=ClientIp
token[1].type=IPAddress
token[2].name=Method
token[2].type=String
token[3].name=URL
token[3].type=String
token[4].name=Status
token[4].type=String
```

Token Declarations

```
event.deviceReceiptTime=Time_of_the_event
event.sourceAddress=ClientIp
event.deviceSeverity=Status
event.requestUrl=URL
event.requestMethod=Method

event.deviceVendor=__getVendor("MyVendor")
event.deviceProduct=__stringConstant("MyProduct")
```

Event Mapping

```
severity.map.veryhigh.if.deviceSeverity=404,500
severity.map.medium.if.deviceSeverity=303,302
severity.map.low.if.deviceSeverity=200..204
```

Severity Mapping

```
extraprocessor.count=1

extraprocessor[0].type=regex
extraprocessor[0].filename=securitymanager/Name-Name
extraprocessor[0].field=event.name
extraprocessor[0].flexagent=false
extraprocessor[0].clearfieldafterparsing=false
```

Extra Processors

Token Declarations

The Token Declarations section specifies the tokens that will be parsed from each input record. Each token has a name and a type. Depending on the type, some tokens (such as TimeStamp) have a format, as well. XML FlexConnectors also have a path expression and a context node, which are described in [“Configuration Properties for an XML FlexConnector”](#).

In addition to assigning parsed tokens to events, you can also assign built-in tokens, which are described in [“Event Mapping”](#).

Parameter	Description
token.count	This property specifies the number of tokens that each line of the file contains. For example, token.count = 7 indicates there are seven tokens. Token declarations are numbered from 0 to token.count-1.

token[x].name	This property specifies a user-defined name for the token, this can be a friendly name used to identify the token. For example, token[0].name=Time_of_the_event would set the name of the token of index 0 to Time_of_the_event. Use this friendly name to identify how to map it to the event object.
token[x].type	This property specifies the data type of the object. It is important to set the correct type so the mapping to the event object can be correctly performed. For a list of supported types, see "Token Types" .
token[x].format	This property modifies the type of the token, for example, when using the Timestamp type, the format defines the actual format of the timestamp. See "Date and Time Format Symbols" .

Token Types

Token types are important because tokens can only be mapped to ArcSight event fields with matching types. See ["ArcSight Built-in Token Types"](#) for descriptions of the token types. They are also listed in the *ArcSight Console User's Guide*, in the *Reference Guide*, under "Data Fields".

Event Mapping

The Event Mapping section lists tokens by name, which are mapped to ArcSight event fields, such as **event.sourceAddress**. The type of the token must match the type of the ArcSight Event field.

In addition to the tokens that are parsed from each input record, you can also configure built-in tokens for specific FlexConnector. Built-in tokens are predefined strings that assign values associated with them to events. For example, if you want to set the **event.deviceHostName** to the name of the syslog sender, you can set **event.deviceHostName=_SYSLOG_SENDER**.

For a complete list of built-in tokens available for each type of FlexConnector, see ["ArcSight Built-in Tokens"](#). For a complete list of the ArcSight event fields, see ["ArcSight Built-in Event Field Mappings"](#).

See ["RequestUrl Event Field"](#) for information on how to use requestUrl.

RequestUrl Event Field

The connector returns a URL when the requestUrl event field is invoked. The URL is stored in the event table. ESM can then parse the URL to derive the following URIs:

- requestProtocol
- requestUrlAuthority
- requestUrlHost
- requestUrlPort

- `requestUrlFileName`
- `requestUrlQuery`

The `requestUrl` event field has the following format:

<protocol>://<authority>@<host>:<port>/<filename>?<query>

Note: Do not set a value for the `requestUrl` event field and set a value for one or more of the URIs.

Setting a value for the `requestUrl` event field and one or more of the URI fields will result in error messages such as: **Attempting to set the `_URL_` when `_URI_` is already set..** or **Attempting to set the `_URIcomponent_` when `_URL_` is already set..** Set values for either the `requestUrl` event field or for one or more of the other URI event fields.

The following two conditions determine the URIs used to create the **`requestUrl`** event field.

1. If the **`requestUrlAuthority`** field is not null, ESM combines these URIs to derive the value.

`requestProtocol`
`requestUrlAuthority`
`requestUrlFileName`
`requestUrlQuery`

2. If the **`requestUrlAuthority`** field is null, ESM combines these URIs to derive the value.

`requestProtocol`
`requestUrlHost`
`requestUrlPort`
`requestUrlFileName`
`requestUrlQuery`

The **`requestUrlAuthority`** event field RFC 2396 has the following format:

<userinfo>@<host>:<port>

Operations Table

Operations are used primarily when tokens are mapped to Micro Focus ArcSight event fields. The following list contains the essential operations. "[ArcSight Operations](#)" describes all of the operations that can be used when tokens are mapped to Micro Focus ArcSight event fields.

IP Address Operations

- `__doubleToAddress`
- `__noDot4QuadStringsToAddress`
- `__numberToAddress`
- `__regexTokenAsAddress`

Number Operations

- `__regexTokenAsInteger`
- `__safeToInteger`
- `__safeToLong`

String Operations

- `__concatenate`
- `__stringConstant`
- `__simpleMap`
- `__toLowerCase`
- `__toUpperCase`

TimeStamp Operations

- `__createLocalTimeStampFromGMTSecondsMillis`
- `__createLocalTimeStampFromSecondsSinceEpoch`
- `__createTimeStamp`
- `__useCurrentYear`

Severity Mapping

The Severity Mapping section provides a severity mapping capability in order to further categorize (or normalize) each event. For example, **`severity.map.low.if.deviceSeverity`**.

FlexConnector severity mapping must be flexible because not all devices will report severity, or use the same format even with devices of the same type. Some use a scale of 0 to 10 levels. Devices that don't really provide a severity-oriented field require that you map severity to an action, or some other event-specific field.

Your severity mappings can also reflect your environment. You might want to consider what would normally be a Medium or Low severity event as Very-High simply because it shouldn't be there to begin with. Or, the opposite: you might lower the severity because the event represents a normal situation on your network. As a general rule, map severity as accurately as possible and use Filters to ignore noise and Rules to respond to specific incidents.

Given the possibilities for Connector Severity mapping mentioned above you should cover all of the possible values of a device severity with a severity map line. All of the mappings follow the same syntax:

`severity.map.agent_severity.if.deviceSeverity=value`

In this case, **`agent_severity`** will be one of very high, high, medium or low and value can either be a comma-separated list of values or use the `".."` notation for ranges of values.

Examples

```
severity.map.veryhigh.if.deviceSeverity=OPEN-INBOUND
severity.map.low.if.deviceSeverity=DROP
severity.map.medium.if.deviceSeverity=OPEN,CLOSE
severity.map.high.if.deviceSeverity=400..599
severity.map.medium.if.deviceSeverity=300..399
severity.map.low.if.deviceSeverity=100..299
```

This table lists severity mappings:

ArcSight Severity	Property
Very High	severity.map.veryhigh.if.deviceSeverity
High	severity.map.high.if.deviceSeverity
Medium	severity.map.medium.if.deviceSeverity
Low	severity.map.low.if.deviceSeverity

These properties cause the ArcSight Severity to be set to a specific level if the Device Severity is one of the values specified. For example:

```
severity.map.veryhigh.if.deviceSeverity=404,500
```

This would cause a Very High severity event when the status of the request was 404 or 500.

```
severity.map.medium.if.deviceSeverity=303,302
```

This would cause a Medium severity event when the status of the request was 303 or 302.

```
severity.map.low.if.deviceSeverity=200..204
```

This would cause a Low severity event when the status of the request was 200, 201, 202, 203, or 204.

Extra Processors

Optional. You can use the extra processor property to chain two configuration files together. This property is useful if you need to use two or more different types of FlexConnectors for the same data. Extra processors are particularly useful when an event has more than one type of data in it and cannot be parsed by a single parser. This property is also referred to as parser linking.

Extra processor definition:

```
extraprocessor.count= <the number of extra processors>
#index start from 0
extraprocessor[<index>].type= <extra processor type>
extraprocessor[<index>].filename= <extra process file name>
```



```
extraprocessor[<index>].<extra processor variable>=  
<extra processor parameter or conditional value>  
...
```

This example illustrates properties that can be added to a time-based database FlexConnector, which cause it to invoke a Regex configuration file for further processing of the **event.message**:

```
extraprocessor.count=1  
extraprocessor[0].type=regex  
extraprocessor[0].filename=netiq/netiq  
extraprocessor[0].field=event.message  
extraprocessor[0].flexagent=true  
extraprocessor[0].clearfieldafterparsing=false
```

One configuration file can link with many other configuration files (by setting the **extraprocessor.count** to a number greater than one). In addition, there is no limit to the number of configuration files, each containing one or more extraprocessor properties that can be chained together.

The following table lists the extra processor types you can specify.

Extra Processor Type	Description
delimited	For any of the delimited parsers
json	For JSON parsers
keyvalue	For key-value parsers
map	For a map file
ntsubparser	For supported Windows application parsers
regex	For any regular expression parsers
standardkeyvalue	For key-value parsers with "=" as the key-value separator and "," as the key value pair separator
xml	For XML parsers

Except for the map extra processor configuration file, all extra processor configuration files should be placed in the **\user\agent\flexagent** folder. The map extra processor file should be placed in **\user\agent\fc** or **\user\agent\aup\fc**. If a map configuration file exists in both the paths, the one in **\user\agent\aup\fc** overrides the one in **\user\agent\fc**.

The following table lists the fields that can be used with an extra processor:

Field Name	Description
field	The value of this field is the input to the extra processor.

flexagent	true or false <ul style="list-style-type: none">• true: The connector uses the parsers in the flexagent directory.• false: The connector uses the parsers in the fcp directory.
clearfieldafterparsing	Clear the input field after completion of parsing.
charencoding	Specifies the type of character encoding.
overrideeventmappings	true or false <ul style="list-style-type: none">• true: Override the mapping event field.• false: Do not override the mapping event field.
conditionfield	Specifies the condition field the extra processor uses.
conditiontype	Specifies how the condition field relates to the condition values.
conditionvalues	Specifies condition values. Use commas to separate multiple values.
Casesensitive	true or false <ul style="list-style-type: none">• True = Use case sensitive parsing.• False = Do not use case sensitive parsing.
concatenatevalues	true or false Applies to key-value parsers. <ul style="list-style-type: none">• If true and there is a duplicate key, do not override the value, but concatenate the values.• If false and there is a duplicate key, override the value.

Key-Value Parsers

Key-value parsers divide log lines into key-value pairs (key=value), extract the key-value pairs into tokens, and then the tokens are mapped to event fields. An example of a key-value log event:

TIME=28/09/11 08:15:00 SRC=194.168.0.12 DST=195.172.0.12 SPT=4236 DPT=80

Key-value parsers are used with keyvalue extra processors and syslog subagents use key-value parsers for secondary processing. The configuration file name for key-value parsers is **vendor.subagent.sdkkeyvaluefilereader.properties**. Key-value parsers have the following properties:

Property	Description
key.delimiter	Regular expression consisting of single character or string that specifies how key value pairs are separated on a log line. For example, key.delimiter=\\s

key.value.delimiter	Regular expression consisting of single character or string that specifies how keys and values are separated into a single key value pair. For example, key.value.delimiter==
key.regex	Regular expression to capture a key. For example, key.regex=([^\s]+)
text.qualifier	Regular expression consisting of a single character or string that specifies how text is separated in a log line. For example, text.qualifier=“
trim.message	true or false - True trims the leading and trailing white spaces of the log line.
trim.tokens	true or false - True trims the leading and trailing white spaces of each token.
trim.keys	true or false - True trims the leading and trailing white spaces of each key.

FlexConnector Creation Wizard for Delimited Log Files

The FlexConnector Creation Wizard is a GUI program that guides you through the process of creating the configuration file for a FlexConnector that read events from comma-delimited or tab-delimited log-files. The file generated by the wizard can be manually edited to include any FlexConnector features or special operations that the wizard does not support. To illustrate how the wizard works, assume that you have a log file named **sample.log** on drive W: that contains the following content:

```
2003-09-23 12:07:57, Customer Zone Accessed,  
38.1.123.206, 192.168.10.100, POST, /search, ?ID=apple, 302  
2003-09-23 12:07:57, Home Page Accessed,  
38.41.123.206, 192.168.10.100, GET, /search, ?ID=candy, 302
```

This is a comma-separated file, so you would select the Log-file FlexConnector.

1. Start the Log-file FlexConnector Wizard by executing the following command from the **ARCSIGHT_HOME/bin** folder:

```
arcsight flexagentwizard
```

The following screen displays:

Welcome to the FlexAgent Creation Wizard. Please select a sample log that you would like to use to create this flex agent and select a name for the configuration file that will be created.

Log Filename ...

FlexAgent Configuration File

Cancel < Previous Next >

2. Enter or browse to the log file you want to parse and enter the name of the configuration file. Click **Next**.
3. The wizard displays the following screen, on which you specify the format of the log-file:

Please specify describe the format of the file using the following parameters:

Delimiter

Other delimiter (not in the list)

Text qualifier

Comment identifier

Trim fields

Contains empty fields

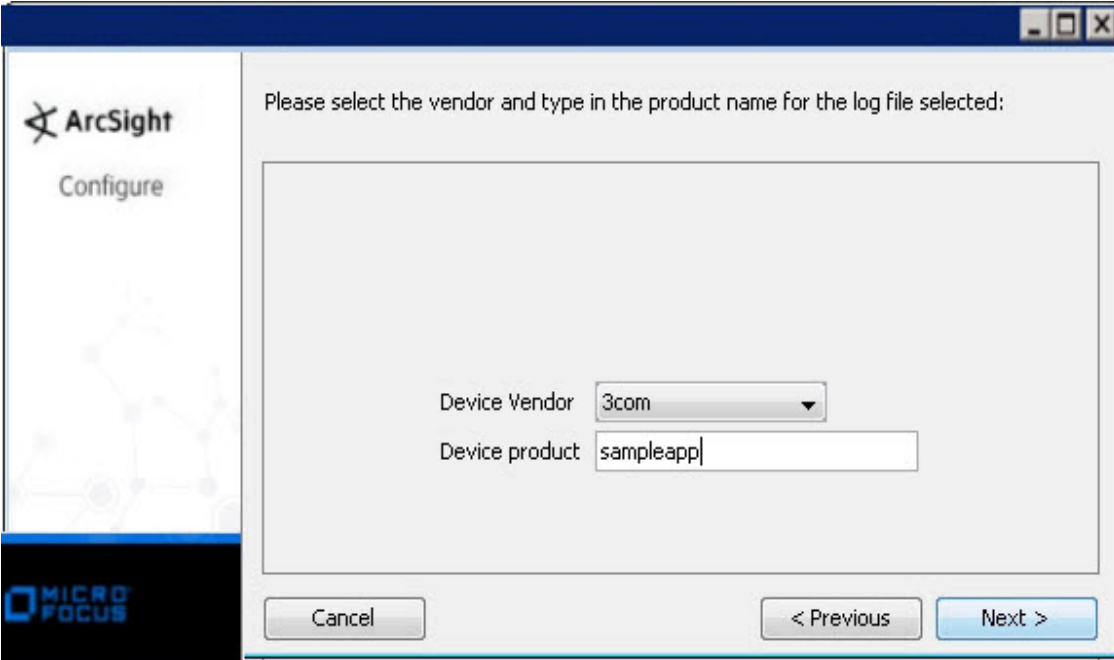
Cancel < Previous Next >

Field	Description
Delimiter	Choose the delimiter that the file is using, in this case ' '
Other delimiter	Use this option if your file contains a delimiter not listed in the "Delimiter" options

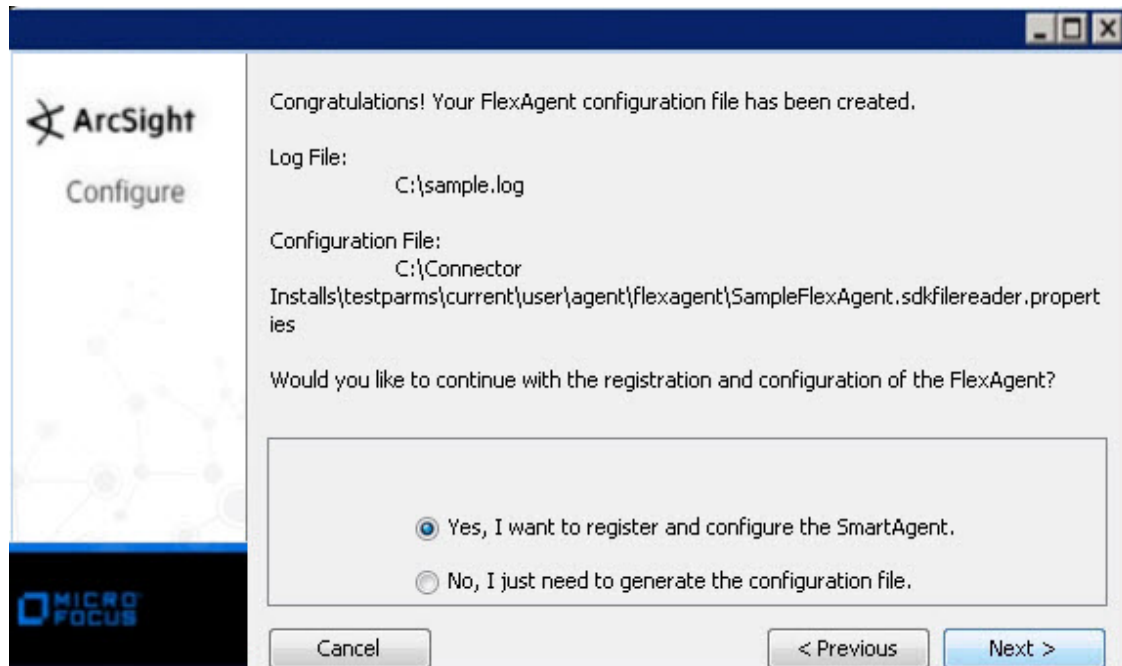
Field	Description
Text qualifier	Sometimes the format contains a character such as a double-quote (") surrounding the text fields. If that is the case, enter that character here. If the character is not found it will be ignored; so for this example, use the default.
Comment identifier	Lines that start with this character will be ignored (the parser will assume that they are comments). For this case, use the default as #.
Trim fields	Set to true if the fields contain leading and/or trailing spaces and you want to remove them from the field
Contains empty fields	Set to true if you are expecting to receive empty tokens. The default (true) will work for most cases.

When you are finished entering parameters, click **Next**.

- The wizard reads the specified log file and displays the field mappings. Map each of the parsed fields to a field in the ArcSight Schema. Click **Next**.
- If some of the fields contain dates, the wizard will prompt you for the correct date format in a separate screen. If the format you need does not appear in the list, choose any format and modify it in the generated configuration file. Choose the format and Click **Next**.
- Select a vendor (or unknown) and specify a product name. If you don't see the vendor for your device, select **Unknown** and then edit the entry manually in the configuration file. Click **Next** to continue.



- The wizard displays the following screen. Click **Next** to finish or to launch the connector configuration wizard.



At this point, the FlexConnector configuration-file has been created, so you can edit it directly to make further changes, if required.

Note: If you choose to continue with registration and configuration of the connector, the wizard will remove any existing connector and launch the FlexConnector configuration wizard again, where you can complete the configuration of your connector with your newly-created FlexConnector log-file configuration file. One benefit of this is that the wizard will make sure that your connector is configured properly with the configuration file that you just created.

Regex Tool for Regex FlexConnectors

The FlexConnector Development Kit includes the FlexConnector Regex Tester (Regex Tool) that analyzes **.log** (event data) files using configuration files (parsers, or **.properties** files), and can also generate regular expressions to use as properties in configuration files that you create.

Use the Regex Tool only with Regex (regular expression) parsers.

See "[Developing a Syslog FlexConnector](#)" for general instructions on using the Regex Tool to create a syslog FlexConnector.

To analyze log files using a parser in the Regex Tool:

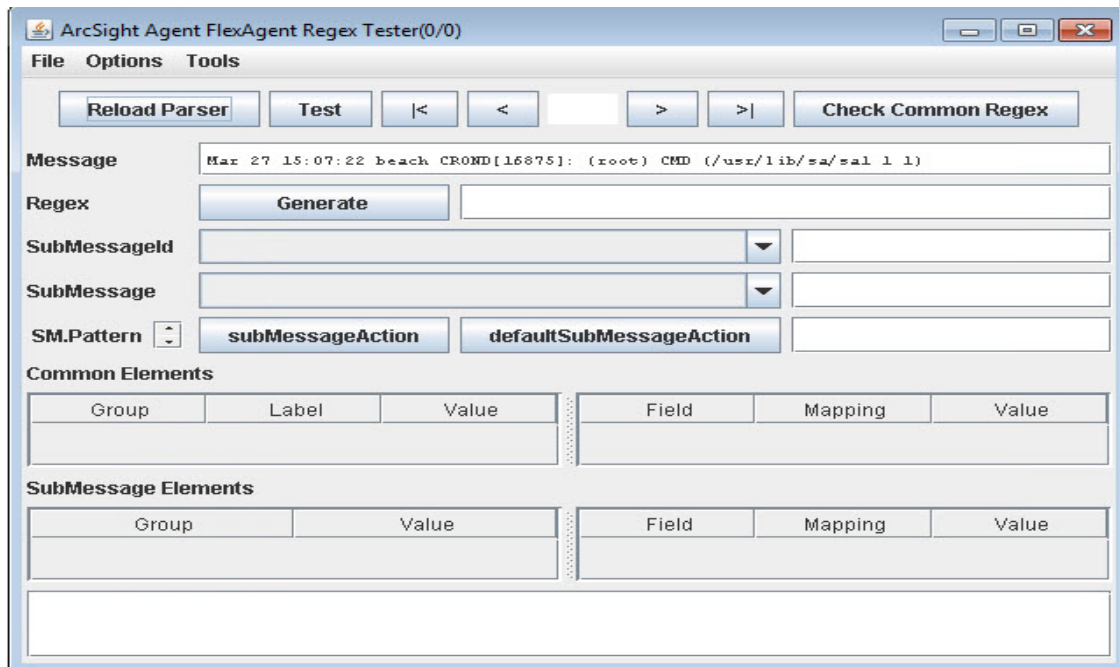
1. Copy the parser file and log file you wish to analyze into this location:

ARCSIGHT_HOME\current\user\agent\flexagent

2. Run the Regex Tool by executing:

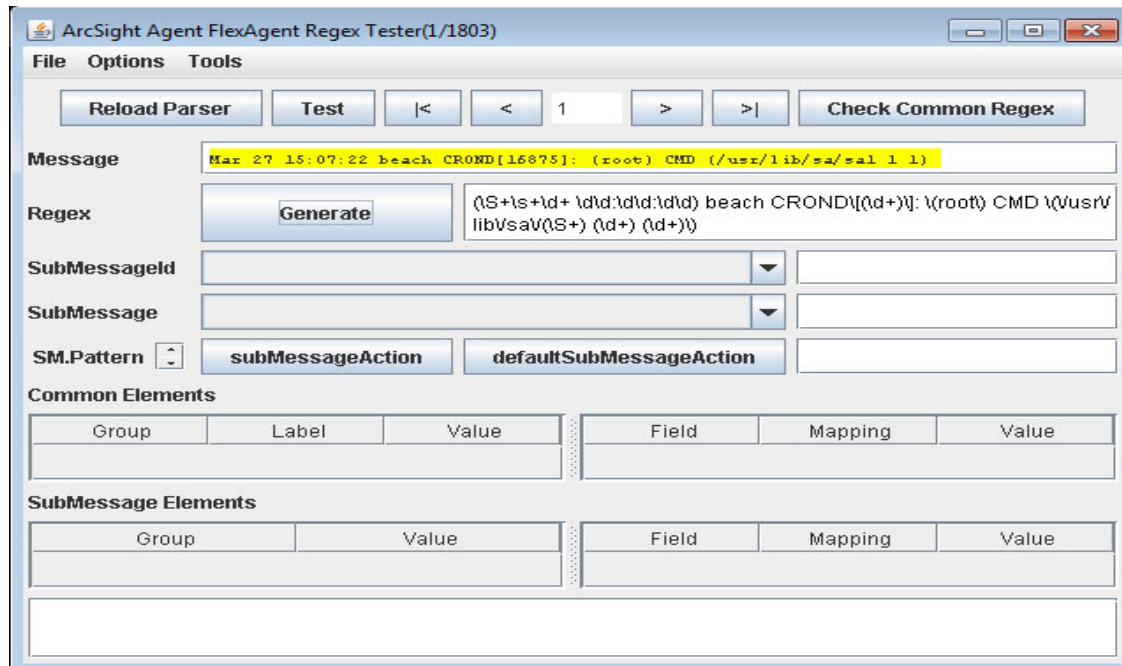
ARCSIGHT_HOME\current\bin\arcsight regex

3. Select **File > Load FlexAgent Regex File** and browse to **ARCSIGHT_HOME\current\user\agent\flexagent** to select and load the parser file (the **.properties** file).
4. Select **File > Load Log File** and browse to **ARCSIGHT_HOME\current\user\agent\flexagent** to select and load the corresponding **.log** file. The first line of the file appears in the Message field, and the number of lines in the file displays on the window title bar.



Also, you can load **.csv** files instead of a **.log** file for analysis. In this case, choose **File > Load CSV Export with Raw Event** rather than **File > Load Log File**. The **.csv** file you load must contain a header as well as the raw event data. Use this feature to parse and test raw events that did not initially parse correctly, and that you have exported to a **.csv** file.

5. If you are working with a syslog connector, select **Options > Treat as Syslog Subagent**. Click the check box to select.
6. Click **Generate** to produce a regular expression that will parse the line shown in the Message field, as shown below:



Notice that literals, such as the square brackets around the date and time, are preserved in the generated regular expression.

Use the navigation buttons to view different lines in the log file.



7. Analyze the log file line by line using the navigation buttons.
8. Select **File > Exit** when data analysis is complete.

When you use the Regex Tool to analyze data, two files are generated:

- **regextester.properties**
- **registrycache.properties**

Delete these generated files when you are done with your data analysis. If you do not delete these files, data will persist in the Regex Tool interface.

To create lines for use in configuration files (parsers):

1. Run the Regex Tool by executing:
ARCSIGHT_HOME\\current\\bin\\arcsight regex
2. Select **File > New FlexAgent Regex File**.
3. Enter a name for the new **.properties** file. This file is generated in the location:

ARCSIGHT_HOME\\current\\user\\agent\\flexagent

The new Regex .properties file is generated containing generic Regex you can use to begin creating a configuration file. This Regex is generated one line at a time, and does not generate an entire parser. The Regex tool lists recommended fields to tokenize and map that are associated with the generated Regex. For example:

Common Elements		
Group	Label	Value
\$0	Message	3
\$1	\$2	212005
\$2	\$3	517
\$3	\$4	100.11.11.2
\$4	\$5	62056

- When you are done, select FlexConnector **File > Save FlexConnector Regex File**.

The Regex tool can also be used to edit existing configuration files by choosing **File > Load FlexConnector Regex File**.

If changes do not work as expected, revert to the previously saved version of the file by clicking **ReloadParser**.

Caution: The Regex tool is designed for single-line use only. You can load the entire log file into the tool, but can only process one event at a time.

- Select **File > Exit** when data analysis is complete.

Start the FlexConnector

Once the FlexConnector is installed and the configuration file is created, start the FlexConnector and test it. Before starting the new connector, make sure that the ArcSight Manager and database or Logger are up and running.

Start the FlexConnector by opening a command window on **ARCSIGHT_HOME/bin** and running:

arcsight agents

For more information about running SmartConnectors, including how to establish a SmartConnector as a service or daemon, refer to the *SmartConnector User's Guide*.

The new FlexConnector should begin sending any events it receives from its device to the ArcSight Manager. In the case of database types, note that only records created after the connector starts will be sent as events.

Chapter 5: Configuration File Examples

The following sections describe examples of configuration files for the various connector types.

- [Configuration Properties for a Log File FlexConnector](#)
- [Configuration Properties for all Regex FlexConnectors](#)
- [Configuration Properties for a Time-based Database FlexConnector](#)
- [Configuration Properties for an ID-based Database FlexConnector](#)
- [Configuration Properties for an SNMP Connector](#)
- [Configuration Properties for an XML FlexConnector](#)
- [Configuration Properties for a JSON Folder Follower FlexConnector/ JSON Multiple Folder Follower FlexConnector](#)
- [JSON Parsers for Complex Event Schemas](#)
- [Configuration Properties for Scanner FlexConnectors](#)

Configuration Properties for a Log File FlexConnector

You can create a configuration properties file for a Log File FlexConnector in two ways:

- Use a text editor to add properties you need.
- Use the FlexConnector Creation Wizard, which is discussed in detail in "[FlexConnector Creation Wizard for Delimited Log Files](#)".

In addition to the properties described earlier, a Log File FlexConnector must also contain Source Log File Format declarations. The Source Log File Format section describes how the FlexConnector will read the source information. The following table lists the properties that you can specify:

Property	Description
comments.start.with	This property specifies which lines of the log file should be ignored and which ones are comments. In this example, you would set this property to a pound sign (#) since every comment begins with this symbol.
contains.empty.tokens	Set this property to "false" only if you are sure that the file being parsed will never contain empty tokens. For example, in the following line: token1,token2,,token4 token3 is empty (there are two commas together), so this flag should be set to true. By default, this flag is set to true.

Property	Description
delimiter	<p>This property specifies which character delimits each of the tokens of the file. In this example, you would set this property to a comma (,) since the tokens are separated by a comma. Other possible values are:</p> <p>delimiter= backslash (\); note that there is a space after the backslash (\)</p> <p>delimiter= pipe ()</p> <p>delimiter= comma (,)</p>
start.at.line	<p>Some files will contain a fixed number of lines as a header before the actual content starts. Using this property you can have the FlexConnector ignore those lines before the actual processing starts. For example, the property:</p> <p>start.at.line=10</p> <p>would ignore the first 9 lines of the file.</p>
text.qualifier	<p>Sometimes the tokens in a file will be surrounded by " or another character (for example, Excel CSV). For example, in the line:</p> <p>"token1", "token2", "token3"</p> <p>All tokens are surrounded by " so you can set this property as:</p> <p>text.qualifier="</p> <p>and the " will not be part of the token value.</p>
trim.message	<p>Removes leading and trailing spaces or tab characters from the full message before sending it to the parser.</p>
trim.tokens	<p>Set this flag to true if you want to remove leading and trailing spaces and tab characters from the token values. By default, this flag is false.</p>

Configuration Properties for all Regex FlexConnectors

For Regex FlexConnectors, the regex property must be set to the regular expression:

```
regex=(.*) ([^\ ]*) ([^\ ]*)\\[\\d+\\]: (.*?) password for (.*?) from  
(\\d+\\.\\d+\\.\\d+\\.\\d+) port (\\d+) ssh2
```

Additionally, you can configure these:

- The **trim.message** and **trim.submessage** properties that trim (remove leading and trailing spaces or tab characters) the full message and sub-message before sending it to the parser.
- Sub-messages that allow a Regex-based FlexConnector to switch intelligently between regular expressions. For more information about sub-messages, see ["Sub-Messages"](#).
- Optional properties in the **agent.properties** file that when configured allow you to control which log files to process in a folder, whether to process the folder and subfolders recursively, and so on. These properties are discussed in [Log Internal Events for File-Reading FlexConnectors](#).

Configuration Properties for a Time-based Database FlexConnector

The following is an example of a time-based Database FlexConnector configuration file:

<pre>version.order=1 version.id=5.0 version.query=SELECT idAlert from AlertView</pre>	Version
<pre>query = \ SELECT \ ComputerName, ComputerDomain, Culprit, DNSName, Name, idAlert, \ Description, RepeatCount, AlertLevel, TimeRaised, TimeOfFirstEvent, \ TimeOfLastEvent, TimeResolved, CustomField1, CustomField2, \ CustomField3, CustomField4, CustomField5 \ FROM \ AlertView \ WHERE \ TimeRaised >= ? \ ORDER BY \ TimeRaised</pre>	Query
<pre>timestamp.field=TimeRaised</pre>	Timestamp
<pre>uniqueid.fields=idAlert</pre>	UniqueID
Token Mapping, Event Mapping, Severity Mapping, ExtraProcessors	

Note: Ensure that queries conform with the schema definition so as to avoid errors such as case sensitivity. For example, if the database fields are using all uppercase, column names in the queries and the values in the **timestamp.field** and the **uniqueid.field** should use uppercase:

timestamp.field=TIME_STAMP

uniqueid.field=UNIQUE_ID

In addition to the common properties listed in "[Parser File Structure](#)", the following properties need to be configured for time-based database FlexConnectors:

Version

Mandatory. The version properties enable you to define the order in which the parser files will be sequentially processed. If there are multiple parser files there should be one for each version of the database with which the FlexConnector communicates.

Note: If you are not concerned about the connector adjusting to new versions, you can skip the version check by doing the following: set **version.order=1** and omit **version.query** and

version.id. Note that this will remove the safeguard of checking the schema version.

- **version.order**—Specifies the order in which versions are checked, from the lowest number to the highest; for example, if you have two parser files parserA and parserB and you want to process parserB before parserA, set parserB's **version.order=1** and parserA's **version.order=2**.
- **version.query**—This property enables you to perform a test query against the database to validate the database version. Specify a unique entity in the database schema that differentiates it from other database versions. For example, **version.query=SELECT idAlert from AlertView**.
- **version.id**—If the **version.query** succeeds, the **deviceVersion** token (described in "[ArcSight Built-in Event Field Mappings](#)") is set to the **version.id**. Typically, you would assign the database version as the value for this property. However, you can assign any integer value. For example, if the product version is 8.1, assign **version.id=8.1**.

Query

Mandatory. This property retrieves the rows that were inserted between the last time the query was run and the current time. The query is executed every five seconds, but the frequency can be configured.

For example:

```
query = \  
SELECT \  
    ComputerName, ComputerDomain, Culprit, DNSName, Name, idAlert,\  
    Description, RepeatCount, AlertLevel, TimeRaised, TimeOfFirstEvent, \  
    TimeOfLastEvent, TimeResolved, CustomField1, CustomField2, \  
    CustomField3, CustomField4, CustomField5 \  
FROM \  
    AlertView \  
WHERE \  
    TimeRaised >= ? \  
ORDER BY \  
    TimeRaised
```

To change the frequency at which the query is executed, set the **agent[x].frequency** property in **ARCSIGHT_HOME\current\user\agent\agent.properties**.

All syntactically and semantically correct SQL statements are supported in SELECT queries with the following exception:

- Only one question mark is supported in a time-based Database FlexConnector query.

Timestamp

Mandatory. Specifies the field to use to determine when to run the next query; for example, for the query specified earlier in this section, you can set the timestamp field to **timestamp.field=TimeRaised**.

UniqueID

Mandatory. Specifies the fields to use to distinguish rows with the same timestamp field; for example, for the query specified earlier in this section, you can set the unique ID field to **uniqueid.fields=idAlert**. Use a comma-separated list to specify multiple values for this field.

Configuration Properties for an ID-based Database FlexConnector

The following is an example of the ID-based Database FlexConnector configuration file:

```
version.order=1
version.id=5.0
version.query=SELECT idAlert from AlertView
```

Version

```
query=SELECT events.summary.eid, hostid, hostid_b, start_time, end_time,
alert_level \
  FROM events.summary, events.host \
  WHERE type=0 and events.summary.eid = events.host.eid and end_time is
not null and end_time > ? \
  ORDER BY end_time
```

Query

```
maxid.query=select max(end_time) from events.summary
```

MaxID

```
id.field=end_time
```

ID

```
uniqueid.fields=eid,start_time,end_time,hostid,hostid_b
```

UniqueID

```
query.limit=5
```

Query Limit
(Optional)

```
Token Mapping, Event Mapping, Severity Mapping, ExtraProcessors
```

Note: Ensure that queries conform with the schema definition so as to avoid errors such as case sensitivity. For example, if the database fields are using all uppercase, the column names in the queries and the values in the **id.field** and the **uniqueid.field** should use uppercase:

id.field=ID

uniqueid.field=UNIQUE_ID

In addition to the common properties listed in "[Parser File Structure](#)", the following properties should be configured for an ID-based database FlexConnectors.

Version

Mandatory. The version properties enable you to define the order in which the parser files will be sequentially processed. If there are multiple parser files there should be one for each version of the database with which the FlexConnector communicates.

Note: If you are not concerned about the connector adjusting to new versions, you can skip the version check by doing the following: set **version.order=1** and omit **version.query** and **version.id**. Note that this will remove the safeguard of checking the schema version.

- **version.order**—Specifies the order in which versions are checked, from the lowest number to the highest; for example, if you have two parser files parserA and parserB and you want to process parserB before parserA, set parserB's **version.order=1** and parserA's **version.order=2**.
- **version.query**—This property enables you to perform a test query against the database to validate the database version. Specify a unique entity in the database schema that differentiates it from other database versions. For example, **version.query=SELECT idAlert from AlertView**.
- **version.id**—If the **version.query** succeeds, the **deviceVersion** token (described in "[ArcSight Built-in Event Field Mappings](#)") is set to the **version.id**. Typically, you would assign the database version to which the configuration file pertains as the value to this property; however, you can assign any integer value. For example, if the product version is 8.1, assign **version.id=8.1**.

MaxID

Mandatory. Specifies the query to use to retrieve the maximum ID present in the database when the query is run; for example, **maxid.query=select max(end_time) from events.summary**.

Query

Mandatory. This property retrieves the rows that were inserted between the last checked ID and the maximum ID (maxid) at the current time. The query is executed every five seconds, but this frequency is configurable.

For example:

```
query=SELECT events.summary.eid, hostid, hostid_b, start_time, end_time, \
alert_level \
FROM events.summary, events.host \
WHERE type=0 and events.summary.eid = events.host.eid and end_time is \
```

```
not null and end_time > ? \
ORDER BY end_time
```

To change the frequency at which the query is executed, set the **agent[x].frequency** property in **ARCSIGHT_HOME\current\user\agent\agent.properties**.

All syntactically and semantically correct SQL statements are supported in SELECT queries.

ID

Mandatory. Specifies the field to use to determine when to run the next query; for example, for the query specified earlier in this section, you can set the ID field to **id.field=end_time**.

UniqueID

Optional. Specifies the field to use to distinguish rows with the same ID field; for example, for the query specified earlier in this section, you can set the unique ID field to **uniqueid.fields=eid,start_time,end_time,hostid,hostid_b**.

Use a comma-separated list to specify multiple values for this field.

Note: The IDs for two events might be identical if the ID field is set to an entity such as a timestamp. For example, if the ID field is set to **end_time**, two events may have the same ID. The Unique ID field is used to distinguish such events.

Query Limit

Optional. Specifies the maximum number of rows to return when a query is run; for example, **query.limit=3**. If default value for **query.limit** is set to unlimited; that is, there is no limit imposed on the number of rows that will be returned when a query is run.

Configuration Properties for an SNMP Connector

The information in this section is applicable to the SmartConnector for SNMP Unified.

You can create one of these types of configuration files:

- one configuration file per trap, with the tokens specified in the same order as the variables (varbinds) that appear in the trap packet, or
- a single configuration file with a **trap.types** property that lists the trap types to capture

Example of token mapping in a configuration file per each trap (sdksnmp.0.snmptrap.properties)

Note: When a trap with the trap type=0 is received, the first varbind (in the trap) would be parsed as the first token variable regardless of the oid. The second varbind is parsed as the second token variable, and so on through the varbinds and token variables.

```
token.count=5
token[0].name=ApplicationId
token[0].type=String

token[1].name=IncidentName
token[1].type=String

token[2].name=IncidentSourceNodeHostname
token[2].type=String

token[3].name=IncidentSourceNodeMgmtAddr
token[3].type=String

token[4].name=IncidentOtherNodeMgmtAddr
token[4].type=String
```

**Example of token mapping in a configuration file for a list of trap types
(sdksnmp.0.sdksnmptrap.properties)**

Note that the trap types must be listed in the **trap.types** variable before the **token.count** in the **sdksnmp.0.sdksnmptrap.properties** configuration file. In this example, this is specified as **trap.types=0,1,2** (0, 1, 2 are the trap types in this example). Also, you must define an **.oid** for each token listed in the parser.

```
trap.types=0,1,2
token.count=4

token[0].name=ApplicationId
token[0].oid=1.3.6.1.4.1.11.2.17.19.2.2.1
token[0].type=String

token[1].name=NmsUrl
token[1].oid=1.3.6.1.4.1.11.2.17.19.2.2.2
token[1].type=String

token[2].name=Reserved1
token[2].oid=1.3.6.1.4.1.11.2.17.19.2.2.3
token[2].type=String

token[3].name=Reserved2
```

```
token[3].oid=1.3.6.1.4.1.11.2.17.19.2.2.4  
token[3].type=String
```

Configuration Properties for an XML FlexConnector

The XML FlexConnector parser builds a tree representation of the XML log file. A root node is at the top of the tree, hop nodes are in between, and trigger nodes are at the bottom (where they generate events). The following is an example of an XML FlexConnector configuration file:

<pre>namespace.count=2 namespace[0].prefix=default namespace[0].uri=http://www.mycompany.com/ids/2007/09/example namespace[1].prefix=ac namespace[0].uri=http://www.yourcompany.com/fds/acfg</pre>	Namespace (Optional)
<pre>hop.node.count=1 hop.node[0].name=host hop.node[0].expression=/audits/audit/hosts/host</pre>	Hop Nodes (Optional)
<pre>trigger.node.expression=\$host/applications/application,\$host/vulnerabilities/ vulnerability</pre>	Trigger Node
<pre>token.count=3 token[0].name=startDate token[0].expression=audits/audit/startDate token[0].node=root token[1].name=endDate token[1].expression=audits/audit/endDate token[1].node=root token[2].name=ip token[2].type=IPAddress token[2].expression=ip token[2].node=host</pre>	Token Mapping
Event Mapping, Severity Mapping, Extraprocessors	
<pre>extraevent.count=3 extraevent[0].filename=example_xml_file/example_xml_file.xml3.uris extraevent[0].name=event1 extraevent[1].filename=example_xml_file/example_xml_file.xml3.openports extraevent[1].name=event2 extraevent[2].filename=example_xml_file/example_xml_file.xml3.vulns extraevent[2].name=event3</pre>	Extra Events (Optional)

In addition to the common properties listed in "[Parser File Structure](#)", the following sections list the optional and mandatory properties for an XML FlexConnector configuration file.

Note: You can also configure optional properties in the **agent.properties** file that when configured allow you to control which log files to process in a folder, whether to process the folder and subfolders recursively, and so on. These properties are discussed in "[Log Internal Events for File-Reading FlexConnectors](#)".

Namespace

Optional. However, if your XML log file uses explicit namespaces or a default namespace, you must specify those namespaces using these properties:

- **namespace.count**—Specifies the number of namespaces that your XML log file uses; for example, **namespace.count=2**.
- **namespace.prefix**—Specifies the namespace prefix to use; for example, **namespace[1].prefix=ac**.
- **namespace[x].prefix=default**—Use when your XML file specifies a namespace but does not use any prefixes in the file. That is, your XML file uses a default namespace.
- **namespace.uri**—Specifies the Uniform Resource Identifier (URI) for the namespace; for example, **namespace[0].uri=http://example.org/2003/08/sdee**

Hop Nodes

Optional. Hop nodes are the nodes in the path from the root node to the event triggering node. These nodes are necessary when tokens need to be captured from nodes other than the triggering node or when events pertaining to a particular node need to be grouped in one block.

Multiple hop node levels can be defined with each new level of hop nodes defined in reference to the previously defined level. Hop nodes can also reference root nodes directly as variables.

To define hop nodes, use these properties:

- **hop.node.count**—Specifies the number of hop nodes; for example, **hop.node.count=1**
- **hop.node.name**—Specifies the names for the hop nodes; for example, **hop.node[0].name=host**
- **hop.node.expression**—Specifies the XPath/XQuery path expressions to select the nodes; for example, **hop.node[1].expression=/audits/audit/hosts/host**

Trigger Nodes

Mandatory. These are the nodes that trigger events. An XPath/XQuery path expression for a trigger node can be the last defined hop node or the root node if no hop nodes are available.

To define trigger nodes, use this property:

trigger.node.expression=\$host/applications/application

Token Mappings

Mandatory. In addition to the token properties listed in "[Token Declarations](#)", you must specify these two properties for the XML parser:

- **token[x].expression**—Specifies the XPath/XQuery path expression that is traversed to obtain the value for the token. This is a mandatory property. For example,
token[0].expression=audits/audit/startDate
- **token[x].node**—Specifies the context node—root node, hop node, or trigger node—relative to which the path expression is evaluated. A context node can be a hop node or a root node. If this property is not specified, it defaults to the trigger node. For example,
token[0].node=host

Examples of Token Mappings

- A token captured from the root node:
token[0].expression=audits/audit/startDate
- A token captured from the hop node 1:
token[2].name=ip
token[2].type=IPAddress
token[2].expression=ip
token[2].node=host
- A token captured from the hop node 2:
token[5].name=protocol
token[5].expression=protocol
token[5].node=vulnref
- A token captured from the trigger node, when **token[x].node** is specified:
token[8].name=name
token[8].expression=name
token[8].node=
- A token captured from the trigger node, when **token[x].node** is not specified:
token[13].name=descr
token[13].expression=description

Extra Events

Optional. If you need your FlexConnector to collect different event types for the same trigger node or from different trigger nodes, you can use this property to specify other XQuery configuration files in the current configuration file.

To specify extra events, use these properties:

- **extraevent.count**—Specifies the number of extra events; for example, **extraevent.count=2**
- **extraevent[x].filename**—Specifies the file name of the additional configuration file that this parser should use; for example, **extraevent[0].filename=ncircle_xml_file/ncircle_xml_file.xml****3.uri**
- **extraevent[x].name**—Specifies a name to associate with the extra events; for example, **extraevent[0].name=/scanner/device/uri/aggregated**

Configuration Properties for a JSON Folder Follower FlexConnector/ JSON Multiple Folder Follower FlexConnector

The JSON Folder Follower FlexConnector parser builds a tree representation of the JSON log file. A root node is at the top of the tree and trigger nodes are at the bottom (where they generate events). There may be multiple root nodes in each file. The following is an example of a JSON Folder Follower FlexConnector/ JSON Multiple Folder Follower FlexConnector configuration file:

```
trigger.node.location=/entries
```

```
token.count=5
```

```
token[0].name=type  
token[0].type=String  
token[0].location=type
```

```
token[1].name=eventId  
token[1].type=String  
token[1].location=event_id
```

```
token[2].name=eventType  
token[2].type=String  
token[2].location=event_type
```

```
token[3].name=sessionId
token[3].type=String
token[3].location=session_id

token[4].name=ipAddress
token[4].type=String
token[4].location=ip_address

additionaldata.enabled=true

event.deviceVendor=__stringConstant("Box")
event.deviceProduct=__stringConstant("Box.net")
event.deviceEventClassId=eventType

event.name=eventType

event.sourceUserName=created_by_name
event.sourceUserId=created_by_user_id
event.sourceHostName=ipAddress

#The code uses event.externalId to get the eventId to persist. Please don't
change this mapping. You may get duplicates if you do that

event.externalId=eventId

event.deviceReceiptTime=__createOptionalTimeStampFromString(created_at,"YYYY-
MM-DDThh:mm:ss.SSSX")

event.fileName=__oneOf(source_item_name,source_folder_name)
event.fileId=__oneOf(source_folder_id,source_item_id)
event.fileType=__oneOf(source_item_type,__ifThenElse(source_folder_
id,,,"folder"))

event.destinationUserName=__oneOf(source_name,source_user_name)
event.destinationUserId=__oneOf(source_id,source_user_id)

event.deviceCustomString1=__oneOf(created_by_login,source_login)
event.deviceCustomString1Label=__stringConstant("Source User Email Address")

event.deviceCustomString2=source_type
event.deviceCustomString2Label=__stringConstant("Source Type")
```

Note: You can also configure optional properties in the **agent.properties** file that when configured allow you to control which log files to process in a folder, whether to process the folder and subfolders recursively, and so on. These properties are discussed in "[Log Internal Events for File-Reading FlexConnectors](#)".

Trigger Node

Mandatory. This is the node that triggers events.

To define trigger nodes, use this property:

trigger.node.location=/entries

Token Location and Mappings

Mandatory. In addition to the token properties listed in "[Token Declarations](#)", you must specify this property for the JSON parser:

token[x].location—Specifies the JSON path expression that is traversed to obtain the value for the token. This is a mandatory property.

For example, **token[2].location=event_type**

Examples of token mappings:

- **token[2].name=eventType**
- **token[2].type=String**
- **token[2].location=event_type**

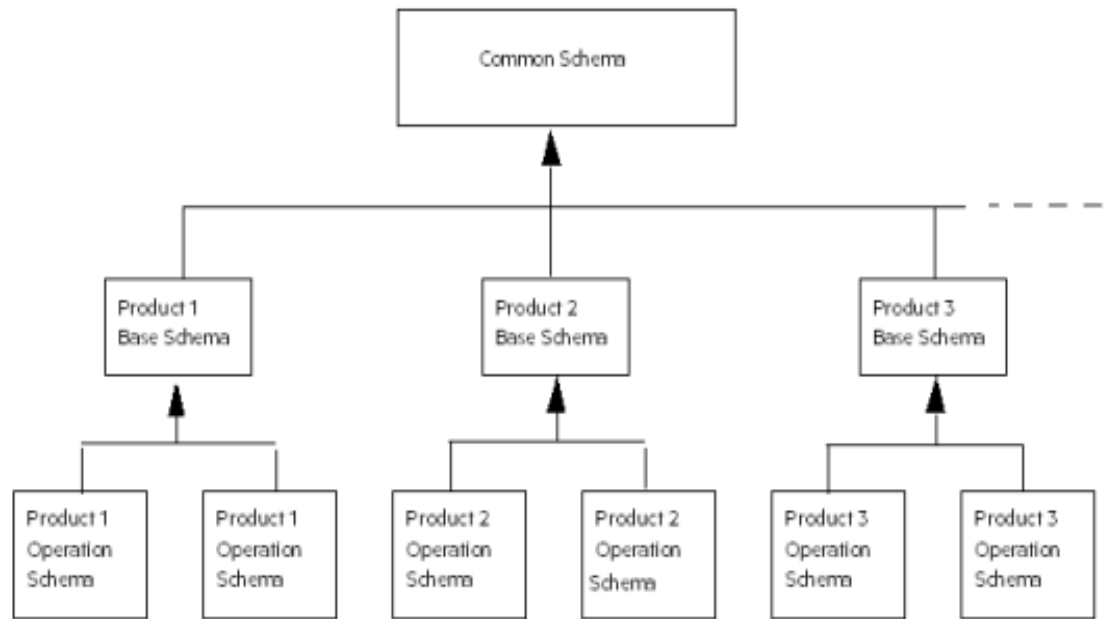
JSON Parsers for Complex Event Schemas

For more complex event schemas, the JSON parser can:

- Handle a hierarchical schema
- Handle an array with a key element
- Represent the value of a token in URI format

Working with Hierarchical Schemas

In some cases, a web application can have a common schema and product-specific schemas, as illustrated in the following figure.



- The product-specific schemas could in turn have product base schemas and more specific schemas.
- The main parser will contain the token specification for the common schema.
- The main parser will have sub-parsers that contain the token specification for the product-specific base schemas.
- Each product base schema sub-parser will, in turn, have its own sub-parsers that contain the token specification for the more specific product schemas.
- Each parser file will have its own event mapping specification.

The `__subParse(mapFileName)` format in the JSON parser supports this scenario.

When the JSON parser encounters this format specified for a token, it looks for the sub-parser. The `mapFileName` points to a map file with a key and a value. The key will be matched with the token value. The value points to the sub-parser file. The sub-parser, if found, will be processed and the resulting **SecurityEvent** will be merged into the parent **SecurityEvent**. An error will be logged if no sub-parser is found. The token itself will still be included in the current token map.

For example, the following common schema has a **RecordType** determining the type of operation, which can be used to determine the sub-parser file:

```
token[5].name=RecordType
token[5].type=string
token[5].format=__subParse(recordtype-map.csv)
token[5].location=RecordType
```

The parameter for `__subParse` is the `recordtype-map` file path, which is a relative path to `$ARCSIGHT_HOME\user\agent\flexagent`.

The main parser is located in `$ARCSIGHT_HOME\user\agent\flexagent`. The sub-parsers (the `.json.properties` files) and the sub-parser map (the `.csv` files) can locate in any sub-folder of that folder.

Also assume that the `recordtype-map.csv` has the following content:

```
1,product_admin
2,product_item
...
```

If the **RecordType** value is **2**, then the `product_item.jsonparser.properties` file will be processed.

Representing a JSON Array with a Key Element

Some event schemas have collections, which are arrays of JSON objects. Some of the collections have a significant element in a JSON object that should be used to identify the rest of the JSON elements.

For example, assume that the value of the **Name** field used in the JSON array illustrated in "[Sample JSON Array](#)" is significant, and its value should be used to identify the element. A desirable output of the token map could be something like the following:

ForwardTo	alias1@mail.com
From	
MoveToFolder	
SentTo	alias2@mail.com

The `__collection(keyField, withPrefix, withPostfix, keyFieldMapFileName)` format in the JSON parser supports this scenario. The arguments to the `__collection` format have the follow definitions:

- **keyField** (required)—points to the significant field.
- **withPrefix** (optional)—indicates if the target token names should be prefixed with the parent key. Default value is **true**.
- **withPostfix** (optional)—indicates if the target token names should be postfixed with the key of the value elements. Default value is **true**.
- **keyFieldMapFileName** (optional)—points to a map file which will be used to map a more meaningful target token name.

The delimiter for the prefix and postfix is `"->"`.

The `__collection` format can be used only on an array node. The array node itself is not part of the current token map. The array is processed to generate a token map, which is merged into the current token map. This format is ignored if it is applied to a non-array node, and the node is handled as normal.

Using the JSON array in Sample JSON Array as an example, the **Parameters** token will be as follows:

```
token[5].name=Parameters
token[5].type=String
token[5].format=__collection(Name,false,false)
token[5].location=Parameters
```

The format value `__collection (Name,false,false)` produces the following token map:

ForwardTo	alias1@mail.com
From	
MoveToFolder	
SentTo	alias2@mail.com

Changing the format value to `__collection(Name)` produces the following token map:

Parameters->ForwardTo->Value	alias1@mail.com
Parameters->From->Value	
Parameters->MoveToFolder->Value	
Parameters->SentTo->Value	alias2@mail.com

Changing the format value to `__collection(Name,,false)` produces the following token map:

Parameters->ForwardTo	alias1@mail.com
Parameters->From	
Parameters->MoveToFolder	
Parameters->SentTo	alias2@mail.com

Representing a Token Value in URI Format

There are times when it is desirable to translate a JSON node into a URI format string. The `__uri()` format in the JSON parser supports this scenario.

This format can be applied to any node.

Using the JSON array in ["Sample JSON Array"](#) as an example, the **Parameters** token will be as follows:

```
token[5].name=Parameters
token[5].type=String
token[5].format=__uri()
token[5].location=Parameters
```

The token specification above produces the following token map:

Parameters	Name:"ForwardTo" /Value:"alias1@mail.com" /Name:"From" /Value:"" /Name:"MoveToFolder" /Value:"" /Name:"SentTo" /Value:"alias2@mail.com"
------------	---

Sample JSON Array

The following code represents a JSON array.

```
"Parameters": [  
  {  
    "Name": "ForwardTo",  
    "Value": "alias1@mail.com"  
  },  
  {  
    "Name": "From",  
    "Value": "" },  
  {  
    "Name": "MoveToFolder",  
    "Value": ""  
  },  
  {  
    "Name": "SentTo",  
    "Value": "alias2@mail.com"  
  }  
]
```

Configuration Properties for Scanner FlexConnectors

The configuration properties you can set depend on the type of source report that the FlexConnector will process. These source report types are described in the following sections:

- ["Scanner FlexConnectors for Normal Text or XML Scan Reports"](#)
- ["Configuration Files for XML Reports"](#)
- ["Scanner FlexConnectors for Database Scan Reports"](#)

Scanner FlexConnectors for Normal Text or XML Scan Reports

Scanner FlexConnectors that process normal text or XML scan reports are parsers that make several passes through the scan report to extract relevant information. The first pass must be to get a list of hosts scanned. Subsequent passes for extracting vulnerability, open ports, operating system, and applications information can be run in any order.

Note: Avoid using // symbols which can have a huge impact on performance. For example, `$root//@startTime` -- scans every node in the entire document for the `startTime` attribute.

To define a scanner FlexConnector for normal text or XML scan reports, you must define parser files to retrieve the following information:

- A list of hosts scanned
- The vulnerabilities for each scanned host
- The open ports for each scanned host
- The operating system and applications running on each scanned host

The name of the configuration files and properties required for each depend on whether the configuration file is for processing a normal text scan report or an XML report.

How Scanner FlexConnectors Parse Scan Reports

A scanner FlexConnector obtains the following information from a scan report:

- List of hosts scanned
- List of open ports
- List of vulnerabilities
- Operating systems and applications on each host
- Any other information such as users, shares, and so on

This information is obtained by making several passes over the report. The first pass obtains a list of hosts while subsequent passes, which can be done in any order, obtain the remaining information.

Scanner FlexConnectors retrieve information from scan reports that provide data in normal text or XML form use multiple parsers to obtain information in which each parser extracts information specific to its function. That is, the first parser extracts the list of hosts and defines the other parsers and the order in which they will run. The subsequent parsers extract the open ports, vulnerabilities, operating system, and applications information for each host.

Scanner FlexConnectors that scan results in a database use one database parser that defines the SQL queries required to extract information from the database. Each SQL query represents a single pass that extracts information specific to its function as described for scanner FlexConnectors for normal text or XML form reports. A few additional SQL queries are also included in this FlexConnector to obtain the version of the database, obtain a list of scan jobs in the database, and so on.

Parser Files for Normal Text Reports

To create parser files for normal text reports, see the following sections:

- ["Getting a List of Hosts "](#)
- ["Getting Vulnerabilities for Scanned Hosts"](#)
- ["Getting Open Ports on Scanned Hosts"](#)
- ["Getting OS and Applications \(URLs\) on Scanned Hosts"](#)

Getting a List of Hosts

The configuration file for getting a list of scanned hosts must be named **vendor.scanner.sdkrfilereader.properties**, where **vendor** is usually the name of the scanner device vendor.

The following is an example configuration file for getting a list of hosts from a scan report:

```
line.include.regex=[\\w\\. -]+\\|.*

regex=(\\w\\. -)+\\|(.*)\\|(.*)\\|.*

token.count=3
token[0].name=ip
token[0].type=IPAddress
token[1].name=hostname
token[2].name=mac
token[2].type=MacAddress

event.destinationAddress=ip
event.destinationHostName=hostname
event.destinationMacAddress=mac

use.ip=false

invalid.vulnerability.ids=CVE|null,CVE|NOCVE,CVE|,Nessus|

extraevent.count=3>
extraevent[0].filename=nessus_nsr_osuris
extraevent[0].name=/scanner/device/uri/aggregated
extraevent[1].filename=nessus_nsr_openports
extraevent[1].name=/scanner/device/openport/aggregated
extraevent[2].filename=nessus_nsr_vulnerabilities
extraevent[2].name=/scanner/device/vulnerability/aggregated
```

In addition to the common properties listed in "[Parser File Structure](#)", the following properties need to be configured:

Ignore or Include Line

Optional. This property enables you to specify filters (as regular expressions) that help identify lines in a scan report that need to be processed to obtain information about scanned hosts. Lines that do not meet the criteria specified in the filter are not processed.

The **include** filter specifies a regular expression that a line must match for it to be processed for extracting scanned hosts.

The **ignore** filter specifies a regular expression that, when matched to a line, that line is excluded and not processed for extracting scanned hosts.

The following is the syntax for the **include** and **ignore** filters:

```
line.include.regex=<regular_expression>  
line.ignore.regex=<regular_expression>
```

For example:

```
line.include.regex=[\\w\\. - ]+\\|.*  
line.ignore.regex=[\\w\\. - ]+\\|.*?\\|.*
```

Regular Expression and Token Mappings

Mandatory. At a minimum, IP address or host name must be extracted from the scan report. In addition, if a MAC address and other information are available, they should also be extracted.

The following regular expression extracts IP address, host name, and MAC address into these tokens:

```
regex=([\\w\\. - ]+)(\\.?(?))\\|\\.?(?)(\\.?(?))\\|.*
```

```
token.count=3
```

```
token[0].name=ip  
token[0].type=IPAddress  
token[1].name=hostname  
token[2].name=mac  
token[2].type=MacAddress
```

```
event.destinationAddress=ip  
event.destinationHostName=hostname  
event.destinationMacAddress=mac
```

Use IP

Optional. Some scanners report only the IP addresses or host names of hosts scanned, while others might report both. The Use IP property indicates whether the scan reports contain IP addresses. When this property is set to false, it indicates that the scan report does not contain IP addresses.

```
use.ip=false
```

If this property is not set, the scanner FlexConnector expects IP addresses in the scan report.

Invalid Vulnerabilities

Optional. This property specifies the vulnerability identifiers (IDs) that the FlexConnector should ignore when processing the scan report. If you want to specify multiple vulnerability IDs, separate them with a pipe (|) character.

The syntax for this property is as follows:

```
invalid.vulnerability.ids=<vulnerability_ids>
```

For example:

```
invalid.vulnerability.ids=CVE|null,CVE|NOCVE,CVE|,Nessus
```

Extra Events

Mandatory. These properties specify the names and locations of other configuration files required for parsing the scan report to extract the vulnerabilities, open ports, operating system, and applications information.

- **extraevent[x].filename**—Specifies the file name of the additional configuration file; for example, **extraevent[0].filename=nessus_nsr_osuris**
- **extraevent[x].name**—Specifies a name to associate with the extra events; for example, **extraevent[0].name=/scanner/device/uri/aggregated**

Although you can specify the extra events in any order, you must use the following event names (**extraevent[x].name**):

- **Vulnerabilities:** **/scanner/device/vulnerability/aggregated**
- **Open ports:** **/scanner/device/openport/aggregated**
- **URIs (for operating system and applications):** **/scanner/device/uri/aggregated**

For example:

```
extraevent.count=3  
extraevent[0].filename=nessus_nsr_osuris  
extraevent[0].name=/scanner/device/uri/aggregated  
  
extraevent[1].filename=nessus_nsr_openports  
extraevent[1].name=/scanner/device/openport/aggregated  
  
extraevent[2].filename=nessus_nsr_vulnerabilities  
extraevent[2].name=/scanner/device/vulnerability/aggregated
```

Getting Vulnerabilities for Scanned Hosts

The configuration file for getting vulnerabilities for the scanned hosts must be named **<vendor>.vulns.sdkrfilereader.properties**, where **vendor** is usually the name of the scanner device vendor. This configuration file is used to extract the following information for the scanned hosts:

- Vulnerabilities as indicated by the vendor vulnerability IDs
- Name, description, risk or severity, solution or recommendation, if available
- External references such as CVE, Bugtraq, and so on
- Any other relevant information that is available

Note: Typically, FlexConnectors look for extra processor configurations in **{ARCSIGHT_HOME}/current/user/agent/flexagent/**. The Normal Text Report Scan FlexConnector is an exception. FlexConnectors will look for these configurations in **{ARCSIGHT_HOME}/current/user/agent/aup/<agents[0].entityid>/fcp/**.

Also, instead of looking for a **<vendor>.vulns.sdkrfilereader.properties** file, FlexConnectors look for a **<vendor>.sdkrfilereader.properties** file.

To work around these problems:

- Copy the vulnerability file from **current/user/agent/flexagent/** to **current/user/agent/aup/<agents[0].entityid>/fcp/**.
- Rename the file from **<vendor>.vulns.sdkrfilereader.properties** to **<vendor>.sdkrfilereader.properties**.

The following is an example configuration file for getting vulnerabilities from a scan report:

```
regex=([\\w\\. - ]+\\|)((\\w+).*(\\d+)?.*?)\\|((\\d+)\\|(.*)\\|(.*)
```

```
token.count=7
```

```
token[0].name=ScannedHostNameOrIp
```

```
token[1].name=ServiceDescription
```

```
token[2].name=ServiceName
```

```
token[3].name=Port
```

```
token[3].type=Integer
```

```
token[4].name=PluginId
```

```
token[5].name=Severity
```

```
token[6].name=Description
```

```
event.destinationHostName=ScannedHostNameOrIp
```

```
event.destinationServiceName=ServiceName
```

```
event.destinationPort=Port
```



```
event.transportProtocol=__regexToken(ServiceDescription,"^.*?/(\\w+).*$")
event.deviceEventClassId=__concatenateDeleting
("Nessus=",NessusID,"#",Name,"#",Risk,"#",INFO,"%CVE=",CVE,"%Bugtraq=",Bugtraq,"%|#=/@")
event.name=__concatenate(ServiceName," - ",Severity)
event.deviceSeverity=Severity
event.message=Description

event.categoryTechnique=__stringConstant("/scanner/device/vulnerability")
event.deviceVendor=__stringConstant(Nessus)
event.deviceProduct=__stringConstant(Nessus)

severity.map.veryhigh.if.deviceSeverity=Security Hole,HOLE
severity.map.high.if.deviceSeverity=Security Warning
severity.map.medium.if.deviceSeverity=Security Note,NOTE,INFO,REPORT
```

In addition to the common properties listed in ["Parser File Structure"](#), the following properties need to be configured.

Token Mappings

Mandatory. At a minimum, IP address or host name must be extracted from the scan report. In addition, if a MAC address and other information are available, they should also be extracted, as shown in the previous example.

Event Mappings

Mandatory. The following event mappings must be defined in the configuration file:

- **event.name**
- **event.deviceSeverity**
- **event.categoryTechnique**

The value for this property must be set to the value shown in the previous example.

```
event.deviceEventClassId = __concatenateDeleting("<vendor_vulnerability_
name>=", vendor_vuln_id, "#", Name, "#", Severity, "#", Description, "%",
"<ref_name1>=", ref_id1,"%',"<ref_name2>=", ref_id2, "%|#=/@")
```

Note: Use `__concatenateDeleting()` instead of `__concatenate()` only if the **Description** field contains characters such as %, |, #, =, @, which are used as delimiters in parsers. For information about `__concatenateDeleting()`, see ["ArcSight Operations"](#).

The value for this property is obtained by concatenating the following vulnerability information (as indicated by the syntax above):

- Vendor vulnerability collection name
For example, **"Nessus="** in the example illustrated previously in this section.
- Vendor vulnerability ID
For example, **NessusID** in the example illustrated previously in this section.
- Vendor vulnerability name
For example, **Name** in the example illustrated previously in this section.
- Risk or severity
For example, **Risk** in the example illustrated previously in this section.
- List of description, recommendation, and remediation (separated by the '#' character)
For example, **INFO** in the example illustrated previously in this section.
- List of external references (separated by the '%' character)
For example, **"%CVE=",CVE,"%Bugtraq=",Bugtraq** in the example illustrated previously in this section.
- **event.destinationHostName**, **event.destinationAddress**, **event.destinationMACAddress**, and **event.destinationPort** (whichever is available)
If you are setting the **event.destinationPort** field, it must contain the open port that the scanner reported.

Severity Mappings

Mandatory. You must define the device severity to FlexConnector severity mapping as shown in section #6 of the example that follows in this section.

Ignore or Include Line

Optional. This property enables you to specify filters (as regular expressions) that help identify lines in a scan report that need to be processed to obtain vulnerability information about scanned hosts. Lines that do not meet the criteria specified in the filter are not processed.

- The **include** filter specifies a regular expression that a line must match for it to be processed for extracting vulnerability information about scanned hosts.
- The **ignore** filter specifies a regular expression that when matches a line, the line is excluded and not processed for extracting vulnerability information.

The syntax for the **include** and **ignore** filters is as follows:

```
line.include.regex=<regular_expression>  
line.ignore.regex=<regular_expression>
```

For example:

```
line.include.regex= [\\w\\. -]+\\|.*?\\|\\d+\\|.*?\\|.*
```

Getting Open Ports on Scanned Hosts

The configuration file for getting the open ports and protocols on each scanned host should be named **vendor.openports.sdkrfilereader.properties**, where **vendor** is usually the name of the scanner device **vendor**.

Note: Typically, FlexConnectors look for extra processor configurations in **{ARCSIGHT_HOME}/current/user/agent/flexagent/**. The Normal Text Report Scan FlexConnector is an exception. FlexConnectors will look for these configurations in **{ARCSIGHT_HOME}/current/user/agent/aup/<agents[0].entityid>/fcp/**.

Also, instead of looking for a **<vendor>.openports.sdkrfilereader.properties** file, FlexConnectors look for a **<vendor>.sdkrfilereader.properties** file.

To work around these problems:

- Copy the vulnerability file from **current/user/agent/flexagent/** to **current/user/agent/aup/<agents[0].entityid>/fcp/**.
- Rename the file from **<vendor>.openports.sdkrfilereader.properties** to **<vendor>.sdkrfilereader.properties**.

The following is an example configuration file for getting an open port on each scanned host from a scan report:

```
regex=([\\w\\. -]+)\\|((.*?) \\((\\d+)/([\\w+)]\\)).*
```

```
token.count=5
```

```
token[0].name=ScannedHostNameOrIp
```

```
token[1].name=Name
```

```
token[2].name=ServiceName
```

```
token[3].name=Port
```

```
token[3].type=Integer
```

```
token[4].name=TransportProtocol
```

```
event.destinationHostName=ScannedHostNameOrIp
```

```
event.name=Name
```

```
event.destinationServiceName=ServiceName
```

```
event.destinationPort=Port
```

```
event.transportProtocol=TransportProtocol
```

```
event.categoryTechnique=__stringConstant("/scanner/device/openport")
```

```
event.deviceVendor=__stringConstant(Nessus)
```

```
event.deviceProduct=__stringConstant(Nessus)
```

In addition to the common properties listed in "[Parser File Structure](#)", the following properties need to be configured:

Token Mappings

Mandatory. At a minimum, the IP address or host name must be extracted from the scan report. In addition, if a MAC address and other information are available, it should also be extracted, as shown in the example above.

Event Mappings

Mandatory. The following event mappings must be defined in the configuration file:

- **event.name**
- **event.categoryTechnique**
The value for this property must be set to the value shown in the previous example.
- **event.transportProtocol**
- **event.destinationPort**
The **event.destinationPort** field must contain the open port that the scanner reported.
- **event.destinationHostName**, **event.destinationAddress**, and **event.destinationMacAddress** (whichever is available)

Ignore or Include Line

Optional. This property enables you to specify filters (as regular expressions) that help identify lines in a scan report that need to be processed to obtain open ports (and protocols) information about scanned hosts. Lines that do not meet the criteria specified in the filter are not processed.

- The **include** filter specifies a regular expression that a line must match for it to be processed for extracting open ports information about scanned hosts.
- The **ignore** filter specifies a regular expression that when matches a line, the line is excluded and not processed for extracting open ports information.

The syntax for the **include** and **ignore** filters is as follows:

```
line.include.regex=<regular_expression>  
line.ignore.regex=<regular_expression>
```

For example:

```
line.include.regex= [\\w\\.\\-]+\\|\\.\\*? \\((\\d+\\/\\w+\\)\\.\\*
```

Getting OS and Applications (URIs) on Scanned Hosts

The configuration file for getting the operating system and applications on each scanned host must be named `<vendor>.uris.sdkrfilereader.properties`, where `vendor` is usually the name of the scanner device vendor.

Note: Typically, FlexConnectors look for extra processor configurations in `{ARCSIGHT_HOME}/current/user/agent/flexagent/`. The Normal Text Report Scan FlexConnector is an exception. FlexConnectors will look for these configurations in `{ARCSIGHT_HOME}/current/user/agent/aup/<agents[0].entityid>/fcp/`.

Also, instead of looking for a `<vendor>.uris.sdkrfilereader.properties` file, FlexConnectors look for a `<vendor>.sdkrfilereader.properties` file.

To work around these problems:

- Copy the vulnerability file from `current/user/agent/flexagent/` to `current/user/agent/aup/<agents[0].entityid>/fcp/`.
- Rename the file from `<vendor>.uris.sdkrfilereader.properties` to `<vendor>.sdkrfilereader.properties`.

The following is an example configuration file for getting operating system and applications on each scanned host from a scan report:

```
regex=([\w\.-]+\|.*?\\|(10336|10785|11936|18261)\\|.*?\\|(.*)
token.count=3
```

```
token[0].name=ScannedHostOrIp
token[1].name=PluginId
token[2].name=Description
```

```
event.destinationHostName=ScannedHostOrIp
event.deviceEventClassId=PluginId
event.message=Description
event.categoryTechnique=__stringConstant("/scanner/device/uri")
event.deviceVendor=__stringConstant(Nessus)
event.deviceProduct=__stringConstant(Nessus)
event.filePath=__getNormalizedOS(OS)
```

Token Mappings

Mandatory. At a minimum, IP address or host name must be extracted from the scan report. In addition, if a MAC address and other information is available, it should also be extracted, as shown in the example above.

Event Mappings

Mandatory. The following event mappings must be defined in the configuration file:

- **event.name**
- **event.categoryTechnique**

The value for this property must be set to the value shown in the previous example.

- **event.filePath**

Use the `__getNormalizedOS()` operation to ensure that the operating system information is translated to a normalized OS asset category, as shown in the example above.

Ignore or Include Line

Optional. This property enables you to specify filters (as regular expressions) that help identify lines in a scan report that need to be processed to obtain the operating system and applications information about scanned hosts. Lines that do not meet the criteria specified in the filter are not processed.

- The **include** filter specifies a regular expression that a line must match for it to be processed for extracting the operating system and applications information about scanned hosts.
- The **ignore** filter specifies a regular expression that when matches a line, the line is excluded and not processed for extracting the operating system and applications information.

The syntax for the **include** and **ignore** filters is as follows:

```
line.include.regex=<regular_expression>  
line.ignore.regex=<regular_expression>
```

For example:

```
line.include.regex= [\\w\\.-]+\\|.*?\\|(? :10336|10785|11936|18261) \\|.*?\\|  
(.*)
```

Configuration Files for XML Reports

Getting a List of Hosts

The configuration file for getting a list of scanned hosts needs to be named **vendor.scanner.xqueryparser.properties**, where vendor is usually the name of the scanner device vendor. In addition, this configuration file specifies the other configuration files to use to extract information and the order in which they need to run.

The following is an example configuration file for getting a list of hosts from a scan report:

```
trigger.node.expression=/report/details/host_info
```

```
token.count=3
```

```
token[0].name=hostname  
token[0].expression=hostname  
token[1].name=ipaddr  
token[1].type=IPAddress  
token[1].expression=ipaddr  
token[2].name=macaddr  
token[2].type=MacAddress  
token[2].expression=macaddr
```

```
event.destinationAddress=ipaddr  
event.destinationHostName=hostname  
event.destinationMacAddress=macaddr
```

In addition to the common properties listed in ["Parser File Structure"](#), the following properties must be configured:

Token Mappings

Mandatory. At a minimum, IP address or host name must be extracted from the scan report. In addition, if a MAC address and other information is available, it should also be extracted, as shown in the example above.

Use IP

Optional. Some scanners report only the IP addresses or host names of hosts scanned, while others might report both. The Use IP property indicates whether the scan reports contain IP addresses. When this property is set to false, it indicates that the scan report does not contain IP addresses.

```
use.ip=false
```

If this property is not set, the scanner FlexConnector expects IP addresses in the scan report.

Invalid Vulnerabilities

Optional. This property specifies the vulnerability identifiers (IDs) that the FlexConnector should ignore when processing the scan report. If you want to specify multiple vulnerability IDs, separate them with a pipe (|) character.

The syntax for this property is:

```
invalid.vulnerability.ids=<vulnerability_ids>
```

For example:

```
invalid.vulnerability.ids=CVE|null,CVE|NOCVE,CVE|,Nessus
```

Extra Events

Mandatory. These properties specify the names and locations of other configuration files required for parsing the scan report to extract the vulnerabilities, open ports, operating system, and applications information.

- **extraevent[x].filename**—Specifies the file name of the additional configuration file; for example, `extraevent[0].filename=nessus_nsr_osuris`
- **extraevent[x].name**—Specifies a name to associate with the extra events; for example, `extraevent[0].name=/scanner/device/uri/aggregated`.

Although you can specify the extra events in any order, you must use the following event names (`extraevent[x].name`):

- **Vulnerabilities:** `/scanner/device/vulnerability/aggregated`
- **Open ports:** `/scanner/device/openport/aggregated`
- **URLs (for operating system and applications):** `/scanner/device/uri/aggregated`

For example:

```
extraevent.count=3
```

```
extraevent[0].filename=saint_xml_file.vulns  
extraevent[0].name=/scanner/device/vulnerability/aggregated
```

```
extraevent[1].filename=saint_xml_file.openports  
extraevent[1].name=/scanner/device/openport/aggregated
```

```
extraevent[2].filename=saint_xml_file.uris  
extraevent[2].name=/scanner/device/uri/aggregated
```

Getting Vulnerabilities for Scanned Hosts

The configuration file for getting vulnerabilities for the scanned hosts needs to be named **vendor.vulns.xqueryparser.properties**, where `vendor` is usually the name of the scanner device `vendor`. This configuration file is used to extract the following information for the scanned hosts:

Vulnerabilities as indicated by the vendor vulnerability IDs

- Name, description, risk or severity, solution or recommendation, if available
- External references such as CVE, Bugtraq, and so on
- Any other relevant information that is available

The following is an example configuration file for getting vulnerabilities from a scan report:


```
hop.node.count=2
```

```
hop.node[0].name=scan_information  
hop.node[0].expression=/report/scan_information
```

```
hop.node[1].name=host_info  
hop.node[1].expression=/report/details/host_info
```

```
trigger.node.expression=$host_info/vulnerability[severity!=  
"Service"]
```

```
token.count=11
```

```
token[0].name=scanner_version  
token[0].expression=$scan_information/scanner_version
```

```
token[1].name=hostname  
token[1].expression=$host_info/hostname
```

```
token[2].name=ipaddr  
token[2].type=IPAddress  
token[2].expression=$host_info/ipaddr
```

```
token[3].name=hosttype  
token[3].expression=$host_info/hosttype
```

```
token[4].name=scan_time  
token[4].type=TimeStamp  
token[4].format=MMM dd HH:mm:ss yyyy  
token[4].expression=$host_info/scan_time
```

```
token[5].name=description  
token[5].expression=description
```

```
token[6].name=severity  
token[6].expression=severity
```

```
token[7].name=cve  
token[7].expression=fn:replace(cve," ","%CVE=")
```

```
token[8].name=impact  
token[8].expression=impact
```

```
token[9].name=resolution  
token[9].expression=resolution
```

```
token[10].name=reference  
token[10].expression=reference
```

```
event.categoryTechnique=__stringConstant("/scanner/device/  
vulnerability")  
event.destinationAddress=ipaddr  
event.destinationHostName=hostname  
event.deviceEventClassId=__concatenate(__concatenateDeleting  
("Saint=",description,"#",description,"#",severity,"#", "Impact",  
impact,"Resolution",resolution," Reference",reference, "%|#=/@"), "%CVE=",cve)  
event.deviceProduct=__stringConstant(SAINT Vulnerability Scanner)  
event.deviceReceiptTime=scan_time  
event.deviceSeverity=severity  
event.deviceVendor=__stringConstant(SAINT)  
event.deviceVersion=scanner_version  
event.name=description
```

```
severity.map.veryhigh.if.deviceSeverity=critical,Critical Problem  
severity.map.high.if.deviceSeverity=concern,Area of Concern  
severity.map.medium.if.deviceSeverity=potential,Potential Problem  
severity.map.low.if.deviceSeverity=info,service,Service
```

In addition to the common properties listed in ["Parser File Structure"](#), the following properties must be configured.

Token Mappings

Mandatory. At a minimum, IP address or host name must be extracted from the scan report. In addition, if a MAC address and other information is available, it should also be extracted, as shown in the example above.

Event Mappings

Mandatory. The following event mappings must be defined in the configuration file:

- **event.name**
- **event.deviceSeverity**
- **event.categoryTechnique**

The value for this property must be set to the value shown in the previous example.

- `event.deviceEventClassId = __concatenateDeleting("<vendor_vulnerability_name>=", vendor_vuln_id, "#", Name, "#", Severity, "#", Description, "%", "<ref_name1>=", ref_id1, "%'", "<ref_name2>=", ref_id2, "%|#=/@")`

Note: Use `__concatenateDeleting()` instead of `__concatenate()` only if the **Description** field contains characters such as %, |, #, =, @, which are used as delimiters in parsers. For information about `__concatenateDeleting()`, see ["ArcSight Operations"](#).

The value for this property is obtained by concatenating the following vulnerability information (as indicated by the syntax above):

- Vendor vulnerability collection name=vendor vulnerability ID
- Vendor vulnerability name
- Risk or severity
- List of description, recommendation, and remediation (separated by the '#' character)
- List of external references (separated by the '%' character)
- `event.destinationHostName`, `event.destinationAddress`, and `destinationMACAddress` (whichever is available)

Severity Mappings

Mandatory. You must define the device severity to FlexConnector severity mapping as shown in section #6 of the example presented earlier in this section.

Getting Open Ports on Scanned Hosts

The configuration file for getting the open ports and protocols on each scanned host needs to be named **vendor.openports.xqueryparser.properties**, where vendor is usually the name of the scanner device vendor.

The following is an example configuration file for getting open port on each scanned host from a scan report:

```
hop.node.count=2
```

```
hop.node[0].name=scan_information
```

```
hop.node[0].expression=/report/scan_information
```

```
hop.node[1].name=host_info
```

```
hop.node[1].expression=/report/details/host_info
```

```
trigger.node.expression=$host_info/vulnerability[severity="Service"]
```

```
token.count=7
```

```
token[0].name=scanner_version
token[0].expression=$scan_information/scanner_version

token[1].name=hostname
token[1].expression=$host_info/hostname

token[2].name=ipaddr
token[2].type=IPAddress
token[2].expression=$host_info/ipaddr

token[3].name=hosttype
token[3].expression=$host_info/hosttype

token[4].name=scan_time
token[4].type=TimeStamp
token[4].format=MMM dd HH:mm:ss yyyy
token[4].expression=$host_info/scan_time

token[5].name=description
token[5].expression=description

token[6].name=severity
token[6].expression=severity

event.applicationProtocol=__regexToken(description,"(?:([a-zA-Z]+) ?)?\\((?
(?:\\d+\\/\\w+)?\\)?")
event.destinationServiceName=__regexToken(description,"(.*) .*")

event.categoryTechnique=__stringConstant("/scanner/device/
openport")

event.destinationAddress=ipaddr
event.destinationHostName=hostname
event.destinationPort=__regexTokenAsInteger(description,"
\\D*(\\d*).?")
event.deviceProduct=__stringConstant(SAINT Vulnerability Scanner)
event.deviceReceiptTime=scan_time
event.deviceSeverity=severity

event.deviceVendor=__stringConstant(SAINT)
event.deviceVersion=scanner_version
```

```
event.name=__concatenate("Service: ",description)
event.transportProtocol=__regexToken(description,"(?:[a-zA-Z]+ ?)?\\(?:
(?:\\d+/(\\w+))?)?\\")
```

In addition to the common properties listed in ["Parser File Structure"](#), the following properties need to be configured:

Token Mappings

Mandatory. At a minimum, IP address or host name must be extracted from the scan report. In addition, if a MAC address and other information is available, it should also be extracted, as shown in the example above.

Event Mappings

Mandatory. The following event mappings must be defined in the configuration file:

- **event.name**
- **event.categoryTechnique**

The value for this property needs to be set to the value shown in the previous example.

- **event.transportProtocol**
- **event.destinationPort**

The **event.destinationPort** field must contain the open port that the scanner reported.

- **event.destinationHostName**, **event.destinationAddress**, and **event.destinationMacAddress** (whichever is available)

Getting OS and Applications (URLs) on Scanned Hosts

The configuration file for getting the operating system and applications on each scanned host needs to be named **vendor.uris.sdkrfilereader.properties**, where vendor is usually the name of the scanner device vendor.

The following is an example configuration file for getting operating system and applications on each scanned host from a scan report:

```
hop.node.count=2
```

```
hop.node[0].name=scan_information
hop.node[0].expression=/report/scan_information
```

```
hop.node[1].name=host_info
hop.node[1].expression=/report/details/host_info
```

```
trigger.node.expression=$host_info

token.count=5

token[0].name=scanner_version
token[0].expression=$scan_information/scanner_version

token[1].name=hostname
token[1].expression=$host_info/hostname

token[2].name=ipaddr
token[2].type=IPAddress
token[2].expression=$host_info/ipaddr

token[3].name=hosttype
token[3].expression=$host_info/hosttype

token[4].name=scan_time
token[4].type=TimeStamp
token[4].format=MMM dd HH:mm:ss yyyy
token[4].expression=$host_info/scan_time

event.categoryTechnique=__stringConstant("/scanner/device/uri")
event.destinationAddress=ipaddr
event.destinationHostName=hostname
event.deviceProduct=__stringConstant(SAINT Vulnerability Scanner)

event.deviceReceiptTime=scan_time
event.deviceVendor=__stringConstant(SAINT)
event.deviceVersion=scanner_version
event.filePath=__getNormalizedOS(hosttype)
event.name=__concatenate("OS: ",hosttype)
```

Token Mappings

Mandatory. At a minimum, IP address or host name must be extracted from the scan report. In addition, if a MAC address and other information is available, it should also be extracted, as shown in the example above.

Event Mappings

Mandatory. The following event mappings must be defined in the configuration file:

- **event.name**
- **event.categoryTechnique**

The value for this property must be set to the value shown in the previous example.

- **event.filePath**

Use the `__getNormalizedOS()` operation to ensure that the operating system information is translated to a normalized OS asset category, as shown in the example above.

Scanner FlexConnectors for Database Scan Reports

Unlike the scanner FlexConnectors that process normal text or XML scan reports, the scanner FlexConnector that processes database scan reports is a single configuration file. This file must be named **vendor.sdkdatabase.properties**, where **vendor** is usually the name of the scanner device vendor.

This file contains properties that extract the following information from a scan report:

- Version of the database
- List of scan jobs stored in the database
- List of hosts scanned in a scan job
- Vulnerabilities, open ports, operating system, and applications for each scanned host in a scan job

Getting the Version of the Database

Version

The following version properties are used to detect and identify the version of the database or product:

- **version.id**
- **version.query**
- **version.order**

For a detailed explanation of these properties, see ["Configuration Properties for a Time-based Database FlexConnector"](#).

Example for FoundScan:

```
version.id=5.x
version.query=select Version from Version where (Name='Database') and
              (Version like '5%')
version.order=3
```

Example for eEye Retina:

```
version.id=5.x
version.order=0
version.query=select count(id_) from eeye_Groups
```

Getting the List of Scan Jobs

Scan Job

The scan job properties obtain a list of scan job IDs for various scan results stored in the database.

- **Query**—Obtains a list of scan job IDs for the scan jobs that have completed.
- **scanjob.column**—Enables you to specify the fields to display in the GUI for scan jobs when the scanner FlexConnector is used in the interactive mode.
- **scanjob.jobid.column.index**, **timestamp.field**, **uniqueid.fields**, and **event.name**—Required by the database parsing framework; therefore, these need to be configured.

For example:

```
query=select jobID,startTime,stopTime,jobDesc from jobs where jobID>? and
termStatus='Finished' order by JobId
scanjob.column.names=jobID,startTime,stopTime,jobDesc
scanjob.column.types=String,String,Integer,TimeStamp,TimeStamp
scanjob.jobid.column.index=3
timestamp.field=stopTime
uniqueid.fields=jobID
event.name=jobDesc
```

Use IP

Optional. Some scanners report only the IP addresses or host names of hosts scanned, while others might report both. The `use.ip` property indicates whether the scan reports contain IP addresses. When this property is set to `false`, it indicates that the scan report does not contain IP addresses.

use.ip=false

If this property is not set, the scanner FlexConnector expects IP addresses in the scan report.

Invalid Vulnerabilities

Optional. This property specifies the vulnerability identifiers (IDs) that the FlexConnector should ignore when processing the scan report. If you want to specify multiple vulnerability IDs, separate them with a pipe (|) character.

The syntax for this property is as follows:

invalid.vulnerability.ids=<vulnerability_ids>

For example:

```
invalid.vulnerability.ids=CVE|null,CVE|NOCVE,CVE|,Nessus
```

Extra Queries

Mandatory. Extra queries are used to extract the list of hosts from a scan job, their open ports, vulnerabilities, operating system and applications on those hosts.

extraevent[x].name—Specifies a name to associate with the extra events; for example, **extraevent[0].name=/scanner/device/uri/aggregated**.

Although you can specify the extra events in any order, you must use the following event names (extraevent[x].name):

- **Vulnerabilities:** /scanner/device/vulnerability/aggregated
- **Open ports:** /scanner/device/openport/aggregated
- **URLs (for operating system and applications):** /scanner/device/uri/aggregated

The extra queries that you must configure are:

- **extra.queries.count**

The number of queries in the configuration file. The number is one less than the total number of queries because the first query starts at 0. For example, if you have three queries defined to extract operating system, open ports, and vulnerabilities, then set this property to 2.

- **last.data.query.index**

The highest index number for the query that will generate events. For example, if you have 6 extra.queries configured and you set this number to 4, any queries with index number 5, 6, and 7 will not generate events; all others will do so.

- **host.query.index**

The index number of the query that generates a list of scanned hosts.

The property **extra.queries.count** determines the number of different queries that will be executed. The order of the extra queries is important. The extra queries that generate events should be placed first, followed by the ones that do not. There are two query index properties: **host.query.index** determines the query that is used to find the hosts in the scan and **last.data.query.index** determines which is the last data query that generates an event that is displayed on the console. The rest of the queries may be used for different purposes, but they do not generate events that are displayed on the console. For example:

```
extra.queries.count=4
last.data.query.index=2
host.query.index=3
```

Vulnerability Query

The vulnerability query extracts the following information:

- Vulnerabilities as indicated by the vendor vulnerability IDs
- Name, description, risk or severity, solution or recommendation, if available
- External references such as CVE, Bugtraq, and so on
- Any other relevant information

This query uses the **order by** clause to sort the results by host ID.

The following is an example of the vulnerability query defined in a configuration file:

```
extra.queries[0].name=/scanner/device/vulnerability/aggregated
```

```
extra.queries[0].query=select
Jobs.JobID,Jobs.EndTime,JobName,Organizations.Name \
      as \
CompanyName,Hosts.IPAddress,OSName,NBName,NBWorkGroup, DNSName,Alive, \
Virtual,ICMP,IdentifyWith,Wireless,Subscan,Batch,VulnFoundID, \
VulnsFound.FaultlineID,CVE,Type,Vulns.Name \
as \
VulnName,Vulns.Description \
as \
VulnDescription,Observation,RiskText,Risk,Recommendation,ExploitDate,Simplici
ty, \
Popularity,Impact,ExploitLink,
Person,LHF,ExploitDataType,Intrusive,SANS,Vulns.Status, \
ScanConfigurations.ConfigurationName \
from \
Jobs,Organizations,Hosts,VulnsFound,Vulns, ScanConfigurations \
where \
Jobs.CustomerID = Organizations.OrgId and \
Jobs.CustomerID = Hosts.CustomerID and Jobs.ConfigurationID =
Hosts.ConfigurationID and \
Jobs.JobID = Hosts.JobID and Jobs.CustomerID = VulnsFound.CustomerID and \
Jobs.ConfigurationID = VulnsFound.ConfigurationID and Jobs.JobID =
VulnsFound.JobId \
and Hosts.HostID = VulnsFound.HostID and VulnsFound.FaultlineID =
Vulns.FaultlineID and \
Jobs.CustomerID = ScanConfigurations.CustomerID and \
Jobs.ConfigurationID = ScanConfigurations.ConfigurationID and Jobs.JobId = ?
\
order by Hosts.HostID

# from Vulns and VulnsFound table
extra.queries[0].event.name=VulnName
```

```
extra.queries[0].event.deviceSeverity=Risk
extra.queries[0].severity.map.high.if.deviceSeverity=7,8,9,10
extra.queries[0].severity.map.medium.if.deviceSeverity=4,5,6
extra.queries[0].severity.map.low.if.deviceSeverity=0,1,2,3
extra.queries[0].event.categoryTechnique=__stringConstant
("/scanner/device/vulnerability")
extra.queries[0].event.deviceEventClassId=__concatenateDeleting
("Faultline=",FaultlineID,"#",VulnName,"#",Risk,"#", "Description",VulnDescrip
tion,"Observation",Observation," RiskText",
RiskText,"Recommendation",Recommendation, "%CVE=",CVE,"%|#=/@")
extra.queries[1].event.destinationAddress=IPAddress
extra.queries[1].event.destinationHostName=DNSName
```

The following event mappings must be defined in the configuration file for the vulnerability query:

- `event.name`
- `event.deviceSeverity`
- `event.categoryTechnique`

The value for this property must be set to the value shown in the previous example.

- `event.deviceEventClassId = __concatenateDeleting("<vendor_vulnerability_name>=", vendor_vuln_id, "#", Name, "#", Severity, "#", Description, "%", "<ref_name1>=", ref_id1,"%',"<ref_name2>=", ref_id2, "%|#=/@")`

Note: Use `__concatenateDeleting()` instead of `__concatenate()` only if the **Description** field contains characters such as %, |, #, =, @, which are used as delimiters in parsers. For information about `__concatenateDeleting()`, see ["ArcSight Operations"](#).

The value for this property is obtained by concatenating the following vulnerability information (as indicated by the syntax above):

- Vendor vulnerability collection name=vendor vulnerability ID
- Vendor vulnerability name
- Risk or severity
- List of description, recommendation, and remediation (separated by the '#' character)
- List of external references (separated by the '%' character)
- `event.destinationHostName`, `event.destinationAddress`, and `destinationMACAddress` (whichever is available)
- Device severity to FlexConnector severity mapping:

```
extra.queries[x].severity.map.high.if.deviceSeverity
extra.queries[x].severity.map.medium.if.deviceSeverity
extra.queries[x].severity.map.low.if.deviceSeverity
```

Open Ports Query

The open ports query extracts the ports open on the scanned hosts and transport protocols allowed on those ports.

This query uses the **order by** clause to sort the results by host ID.

The following is an example of the open ports query defined in a configuration file:

```
extra.queries[1].name=/scanner/device/openport/aggregated
extra.queries[1].query=select
Jobs.JobID,Jobs.EndTime,JobName,Organizations.Name as \
CompanyName,Hosts.IPAddress,OSName,NBName,NBWorkGroup, \
DNSName,Alive,Virtual,ICMP,IdentifyWith,Wireless,Subscan,Batch,ServicesFound.
Banner, \
ServiceName,Services.Port,Protocol,Services.Description, \
Detail, ServicesFound.ServiceID \
from \
Jobs,Hosts,Organizations,ServicesFound,Services where \
Jobs.CustomerID = Organizations.OrgId and \>
Jobs.CustomerID = Hosts.CustomerID and Jobs.ConfigurationID =
Hosts.ConfigurationID and \
Jobs.JobID = Hosts.JobID and \
Jobs.CustomerID = ServicesFound.CustomerID and Jobs.ConfigurationID =
ServicesFound.ConfigurationID \
and Jobs.JobID = ServicesFound.JobId and Hosts.HostID = ServicesFound.HostID
and \
ServicesFound.ServiceID = Services.ServiceID and Jobs.JobId = ? \
order by Hosts.HostID

extra.queries[1].event.name=Service
extra.queries[1].event.destinationPort=Port
extra.queries[1].event.transportProtocol=Protocol
extra.queries[1].event.destinationAddress=IPAddress
extra.queries[1].event.destinationHostName=DNSName
```

The following event mappings must be defined in the configuration file:

- **event.name**
- **event.categoryTechnique**
The value for this property needs to be set to the value shown in the previous example.
- **event.transportProtocol**
- **event.destinationPort**

The **event.destinationPort** field must contain the open port that the scanner reported.

- **event.destinationHostName**, **event.destinationAddress**, and **destinationMacAddress** (whichever is available)

Getting OS and Applications (URLs) on Scanned Hosts

The OS and applications (URLs) query extracts the operating systems and applications found on the scanned hosts.

This query uses the order by clause to sort the results by host ID.

The following is an example of the OS and applications query defined in a configuration file:

```
extra.queries[2].name=/scanner/device/uri/aggregated
extra.queries[2].query=Select IPAddress, DNSName, NBWorkGroup, \
OSName, EndTime from Hosts, Jobs \
where \
Jobs.CustomerID=Hosts.CustomerID and
Jobs.ConfigurationID=Hosts.ConfigurationID and \
Jobs.JobID=Hosts.JobID and Jobs.JobID=?
```

```
extra.queries[2].event.name=OSName
extra.queries[2].event.categoryTechnique=__stringConstant("/
scanner/device/uri")
extra.queries[2].event.filePath=__getNormalizedOS(OSName)
extra.queries[2].event.destinationAddress=IPAddress
extra.queries[2].event.destinationHostName=DNSName
extra.queries[2].event.destinationNtDomain=NBWorkGroup
```

The following event mappings must be defined in the configuration file:

- **event.name**
- **event.categoryTechnique**

The value for this property needs to be set to the value shown in the previous example.

- **event.filePath**

Use the **__getNormalizedOS()** operation to ensure that the operating system information is translated to a normalized OS asset category, as shown in the example above.

Getting Scanned Hosts (Host Query)

This query extracts the IP addresses, host names, MAC addresses of the hosts in a scan job. Because all scanners do not provide all three pieces of information, the query extracts whatever information is available.

This query does not generate events, but generates a list of hosts that the connector uses to create assets and update their information in ESM.

The following is an example of the query:

```
extra.queries[3].name=HostList
extra.queries[3].query= \
SELECT  DISTINCT Hosts.IPAddress, Hosts.DNSName, ServicesFound.Banner \
FROM    Jobs INNER JOIN Hosts ON Jobs.CustomerID = Hosts.CustomerID AND
Jobs.ConfigurationID \
= Hosts.ConfigurationID AND Jobs.JobID = Hosts.JobID \
LEFT OUTER JOIN ServicesFound ON Hosts.JobID = ServicesFound.JobID AND
Hosts.HostID \
= ServicesFound.HostID AND ServicesFound.ServiceID = 236 \
WHERE   Jobs.JobID = ?

>extra.queries[3].event.destinationAddress=IPAddress
extra.queries[3].event.destinationHostName=DNSName
extra.queries[3].event.destinationMacAddress=__getLongMACAddressByString(__
regexToken(Banner,"(?s)MAC Address:\\s*(\\S+)"))
```

Chapter 6: Advanced Features

This chapter contains the following information:

- [Regular Expressions](#)
- [Sub-Messages](#)
- [Log Rotation Types](#)
- [Log Internal Events for File-Reading FlexConnectors](#)
- [Unparsed Events Detection](#)

Regular Expressions

Regular expression-based FlexConnector parse fields from a line-based text log file. The Regex FlexConnector will not manipulate binary files or text files that aren't line based. Multiple line based regex parsers are addressed later in this document.

This table lists meta-characters:

MChar	Definition	Pattern	Sample Matches
.	Any character (except \n new-line)	a.c	abc, aac, acc, adc, aec, ...
	Alternation.	bill ted	ed, bill
{...}	Explicit quantifier notation.	ab{2}c	abbc
[...]	Explicit set of characters to match.	a[bB]c	abc, aBc
(...)	Logical grouping of part of an expression.	(abc){2}	abcabc
*	0 or more of previous expression.	ab*c	ac, abc, abbc, abbbc, ...
+	1 or more of previous expression.	ab+c	abc, abbc, abbbc, ...
?	0 or 1 of previous expression; also forces minimal matching when an expression might match several strings within a search string.	ab?c	ac, abc
\	Preceding one of the above, it makes it a literal instead of a special character. Preceding a special matching character, see below.	a\sc	a c

This table lists escape characters:

Escaped Char	Description
ordinary characters	Characters other than . \$ ^ [()] * + ? \ match themselves.
\t	Matches a tab \u0009.
\r	Matches a carriage return \u000D.
\n	Matches a new line \u000A.
\x20	Matches an ASCII character using hexadecimal representation (exactly two digits).
*	When followed by a character that is not recognized as an escaped character, matches that character. For example, * is the same as \x2A.

This table lists character classes:

Char Class	Description
[aeiou]	Matches any single character included in the specified set of characters.
[^aeiou]	Matches any single character not in the specified set of characters.
[0-9a-fA-F]	Use of a hyphen (–) allows specification of contiguous character ranges.
\w	Matches any word character.
\W	Matches any non-word character.
\s	Matches any white-space character.
\S	Matches any non-white-space character.
\d	Matches any decimal digit. Equivalent to [0-9]
\D	Matches any non-digit. Equivalent to [^0-9]

This table lists common Regex :

Data Type	Regex for FlexConnector	Example
IPAddress	(\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}\\.\\d{1,3})	123.45.67.89
IPAddress:Port	(\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}\\.\\d{1,3})\\:(\\d{1,5})	123.45.67.89:25
Date & Time HTTP	\\[(\\d{2}\\/(\\w+\\/\\d{4}:\\d{2}:\\d{2}:\\d{2} [+-]\\d{4})\\)]	[04/Dec/2004:00:21:37 +0000]
Date & Time	(\\d{2}\\/(\\d{2}\\/(\\d{4} \\d{2}:\\d{2}:\\d{2}) 01/31/2005 10:45:50	31/01/2005 22:15:10

Multi-line Parsing

Some files may contain events that are split into multiple lines. Some types parse files in which each line is an event, but FlexConnector Regex Log file FlexConnectors also support reading multi-line files.

FlexConnectors will try to concatenate all the lines belonging to a single event separated by a space. The problem becomes simpler because the events go back to being one line.

When events are split across several lines, there is typically a way to identify the message start and end. To support multi-line messages, you need to define the message start and end in the configuration file. The properties, in the following table, can be used for this purpose.

This table lists multi-line properties:

Property	Description
multiline.starts.regex	This property can be set to a regular expression that identifies when the multi-line event starts. (This is required for multi-line files.)
multiline.ends.regex	This property can be set to a regular expression that identifies when the multi-line event ends. (This property is optional. If it is not present, it is assumed that when a new event begins the previous one has ended.)
multiline.max.count	This is an overflow protection that is not required but is recommended. The FlexConnector will truncate the message if it reaches this specified number of lines plus one.
multiline.delimiter	By default, lines are concatenated with a space (') between, but this can be changed by setting this property to a different character.
multiline.singleline. nowaiting=(True or False)	If True, the connector does not wait for another line when the log file has a single line without a second line. It proceeds to the next multi-line and continues processing.

Note: Multi-line regular expression support is available only for Log File agents.

A log file that requires a multi-line FlexConnector might look like this:

```
|01/01/2005 11:00:50|1.1.1.1|7663|2.2.2.2|80|this  
is  
a  
message  
that  
takes  
multiple  
lines|  
|01/01/2005 11:00:51|1.1.1.1|7663|2.2.2.2|80|this  
is another large message that takes  
multiple lines|
```

To parse this message with a simple FlexConnector Regex Log file, add the following **multiline** property to the configuration file:

```
multiline.starts.regex=\\|\\d+\\/\\d+\\/\\d+ \\d+:\\d+:\\d+\\|.*
```

The FlexConnector will concatenate multiple lines into a single line. The events will look like this:

```
|01/01/2005 11:00:50|1.1.1.1|7663|2.2.2.2|80|this is a message that takes  
multiple lines|  
|01/01/2005 11:00:51|1.1.1.1|7663|2.2.2.2|80|this is another large message  
that takes multiple lines|
```

Such a log can be parsed by a standard FlexConnector Regex Log file. Another example:

```
multiline.ends.regex=.*\|$
```

In this case, the **ends** property is not required because an expression was defined that will always match the start of a message.

The full FlexConnector Regex Logfile configuration file that can parse this message looks like this:

```
# FlexConnector Regex Configuration File
```

```
multiline.starts.regex=\\|\\d+\\/\\d+\\/\\d+ \\d+\\:\\d+\\:\\d+\\|.*
```

```
regex=\\| (.*)\\| (\\S+)\\| (\\d+)\\| (\\S+)\\| (\\d+)\\| (.*)\\|
```

```
token.count=6
token[0].name=Timestamp
token[0].type=TimeStamp
token[0].format=MM/dd/yyyy HH:mm:ss
```

```
token[1].name=SourceAddress
token[1].type=IPAddress
```

```
token[2].name=SourcePort
token[2].type=Integer
```

```
token[3].name=DestinationAddress
token[3].type=IPAddress
```

```
token[4].name=DestinationPort
token[4].type=Integer
```

```
token[5].name=Message
token[5].type=String
```

```
#submessage.messageid.token=  
#submessage.token=
```

```
event.sourceAddress=SourceAddress  
event.destinationAddress=DestinationAddress  
event.sourcePort=SourcePort  
event.destinationPort=DestinationPort  
event.deviceVendor=__getVendor("MyVendor")  
event.message=Message  
event.deviceProduct=__stringConstant("MyProduct")
```

This is an example of a log file that requires multi-line processing:

Multi-Line Virus Wall Log File

```
Date: 11/29/2004 09:44:11  
Method: HTTP  
From: http://www.nextern.net/downloads/pgtaff/pgtaff.cab  
To: 10.0.1.19  
File: pgtaff.cab  
Action: The uncleanable file is deleted.  
Virus: ADW_SCANPORTAL.A  
-----  
Date: 11/29/2004 11:34:37  
Method: HTTP  
From: http://www.nextern.net/downloads/pgtaff/pgtaff.cab  
To: 10.0.1.19  
File: pgtaff.cab  
Action: The uncleanable file is deleted.  
Virus: ADW_SCANPORTAL.A  
-----  
Date: 11/29/2004 12:21:32  
Method: HTTP  
From: http://192.168.176.227/webplugin.cab  
To: 10.0.1.9  
File: webplugin.cab  
Action: The uncleanable file is deleted.  
Virus: TROJ_ONECLICK.A  
-----
```

The regular expression portion of the configuration file that defines this "Virus Wall" FlexConnector looks like this:

```
multiline.starts.regex=Date:.*  
multiline.ends.regex=-----
```

```
regex=Date:\\s(\\d{2}\\/\\d{2}\\/\\d{4} \\d{2}:\\d{2}:\\d{2}).Method:\\s  
(\\w+).From:\\s(\\S+).To:\\s(\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}\\.\\d  
{1,3}).File:\\s(\\S+).Action:\\s([^\n.]+)\\.\\.Virus:\\s(\\S+).*
```

The following is another example of a multi-line log file:

BEA WebLogic Log File

```
####<30-mar-04 9:04:34 PST> <Info> <HTTP> <bcnproo41> <myserver>  
<ExecuteThread: '9' for queue: 'default'> <> <> <101047> <  
[WebAppServletContext(206735,Scort,/Scort)] SmartDemo  
TerminalNewSessionServlet: web application context path=[/Scort]>  
####<30-mar-04 9:10:35 PST> <Info> <HTTP> <bcnproo41> <myserver>  
<ExecuteThread: '7' for queue: 'default'> <> <> <101047> <  
[WebAppServletContext(206735,Scort,/Scort)] [2004.03.30 09:10:35.468]:Thread  
Group for Queue: 'default'.ExecuteThread: '7' for queue:  
'default'@565cd0:aa4:[Apn2T1cX5PWUVbP4nVXhHM6U714NKT2vVLXhPid1eYtCWY602fn4!-  
1600671479!169410899!3001!7002!1080666038968] !!!  
com.scort.agent.terminal.servlet.ServletHelper.traceOnError Timeout on  
receiving response  
    receive timeout=60000  
    mode=send and receive  
>  
  
####<30-mar-04 9:10:57 PST> <Info> <HTTP> <bcnproo41> <myserver>  
<ExecuteThread: '9' for queue: 'default'> <> <> <101047> <  
[WebAppServletContext(206735,Scort,/Scort)] SmartDemo  
TerminalNewSessionServlet: web application context path=[/Scort]>  
####<30-mar-04 9:11:24 PST> <Info> <HTTP> <bcnproo41> <myserver>  
<ExecuteThread: '9' for queue: 'default'> <> <> <101047> <  
[WebAppServletContext(206735,Scort,/Scort)]
```

In this case, the regular expression portion of the configuration file looks like this:

```
multiline.starts.regex=####<.*  
multiline.ends.regex=####  
regex=####<([>]*)> <([>]*)> <([>]*)> <([>]*)> <([>]*)> <([>]*)> <  
([>]*)> <([>]*)> <([>]*)> <([>]*)>(<.*>
```

Sub-Messages

In some cases, the files being parsed by FlexConnectors may contain more than one message format. For example:

```
Nov 28 22:02:42 10.0.111.2 %PIX-6-106015: Deny TCP (no connection) from
3.3.3.3/4532 to 4.4.4.4/80 flags RST on interface outside
Nov 28 22:06:10 10.0.111.2 %PIX-3-305005: No translation group found for tcp
src inside:10.0.112.9/37 dst outside:4.5.6.7/3562
Nov 29 01:46:42 10.0.111.2 %PIX-6-305005: Translation built for gaddr 1.2.3.4
to laddr 10.0.111.9
Nov 29 01:35:15 10.0.111.2 %PIX-4-500004: Invalid transport field for
protocol=6, from 2.2.2.2/0 to 3.3.3.3/0
Nov 28 12:03:21 10.0.111.2 %PIX-6-106015: Deny TCP (no connection) from
1.1.1.1/3564 to 2.2.2.2/80 flags RST on interface outside
Nov 29 04:11:32 10.0.111.2 %PIX-4-500004: Invalid transport field for
protocol=6, from 5.5.5.5/0 to 6.6.6.6/0
```

There is no easy way to define a regular expression that could match all four possible formats in this example. For this reason, the FlexConnector Regex log-file supports using multiple regular expressions, one for each format, by defining sub-messages.

Almost every message can be divided in two portions, one that is common to all messages and one that varies with each message format. The common, or standard, portion requires only one regular expression. A sub-message is defined as the non-standard portion of the message being parsed. In the example above, divide the message:

```
Nov 28 22:02:42 10.0.111.2 %PIX-6-106015: Deny TCP (no connection) from
199.248.65.116/3564 to 10.0.111.22/80 flags RST on interface outside
```

Into:

```
Nov 28 22:02:42 10.0.111.2 %PIX-6-106015:
```

And:

```
Deny TCP (no connection) from 199.248.65.116/3564 to 10.0.111.22/80 flags RST
on interface outside
```

Identify that the first portion of the message is common to all messages; it contains the month, the day of the month, the time, an IP address and an identifier (in this case formed by the mnemonic **%PIX** (which comes from a Cisco Pix device) followed by a single digit that specifies the device severity and finally message ID). The second portion of the message varies between each of the messages in the example.

Usually one or more sub-messages can be identified by a message ID or format identifier. A message ID, if available, will improve performance of the FlexConnector engine.

In the example, the message ID is the last portion of the identifier provided with each message. Determine that all messages with message ID 106015 have the same format; likewise message identifier 500004. Messages with message ID 305005 have slightly different formats, but they both refer to translations.

The file described above must be parsed using a FlexConnector Regex Log file and sub-messages. The first thing to do is to define a regular expression that will match all messages using knowledge of the standard and non-standard part of the messages:

```
regex=(\S+ \d+ \d+:\d+:\d+) (\S+) %PIX-(\d)-(\d+): (.*)
```

This regular expression matches all the messages above and separates the standard part of the message into tokens. The last `(.*)` matches everything after the `%PIX` identifier; that expression group will become the sub-message to parse further.

With a common expression for all messages, now define the common tokens that are captured:

```
token.count=5
```

```
token[0].name=Timestamp  
token[0].type=TimeStamp  
token[0].format=MMM dd HH\:mm\:ss
```

```
token[1].name=PixIP  
token[1].type=IPAddress
```

```
token[2].name=PixSeverity  
token[2].type=String
```

```
token[3].name=SubmessageIdToken  
token[3].type=String
```

```
token[4].name=SubmessageToken  
token[4].type=String
```

Now add the common mappings:

```
event.deviceReceiptTime=Timestamp  
event.deviceAddress=PixIp  
event.deviceSeverity=PixSeverity  
event.deviceEventClassId=SubmessageIdToken  
event.deviceVendor=__getVendor("CISCO")  
event.deviceProduct=__stringConstant("PIX")
```

Notice that the timestamp does not contain the year (this is typical in a syslog message). Use a FlexConnector operation to add the current year to avoid all of the messages defaulting to the year 1970. The operation to use is `__useCurrentYear()`. The corrected mappings should be:

```
event.deviceReceiptTime=__useCurrentYear(Timestamp)  
event.deviceAddress=PixIp
```

```
event.deviceSeverity=PixSeverity
event.deviceEventClassId=SubmessageIdToken
event.deviceVendor=__getVendor("CISCO")
event.deviceProduct=__stringConstant("PIX")
```

Map the severity, which is also common:

```
severity.map.veryhigh.if.deviceSeverity=0,1
severity.map.high.if.deviceSeverity=2,3
severity.map.medium.if.deviceSeverity=4,5
severity.map.low.if.deviceSeverity=6,7
```

Having parsed the standard part of the message, define which token will contain the message ID and which token will contain the sub-message to be parsed. This is accomplished by defining the following properties:

- **submessage.messageid.token=SubmessageIdToken**
- **submessage.token=SubmessageToken**

The (**submessage.messageid.token**) property identifies the token that will hold the message identifier. The (**submessage.token**) property identifies the token that contains the actual sub-message.

Now define the additional regular expressions for each sub-message ID. To do this, define the number of sub-messages that are required. In this case, there are three sub-message IDs (**106015**, **305005**, **500004**). Define the sub-message count as 3:

```
submessage.count=3
```

Follow these steps to define the sub-message:

1. Define the corresponding sub-message ID.
2. Define the regular expression(s) to use.
3. Define the mappings to event fields.

To define the first sub-message for message ID 106015, first define the message ID:

```
submessage[0].messageid=106015
```

Next, define the number of regular expressions (also known as patterns) needed. Message 305005 will require two regular expressions but the other messages will require only one:

```
submessage[0].pattern.count=1
```

Define the regular expression to use for this message ID:

```
submessage[0].pattern[0].regex=Deny (\\S+) \\(no connection\\) from
(\\d+\\.\\d+\\.\\d+\\.\\d+)/ (\\d+) to (\\d+\\.\\d+\\.\\d+\\.\\d+)/ (\\d+)
flags RST on interface (\\S+)
```

The expression captures the protocol, the source address and source port, the destination address and destination port and finally the interface. Now define how these tokens will map into event fields:

```
submessage[0].pattern[0].fields=event.transportProtocol,  
event.sourceAddress,event.sourcePort,  
event.destinationAddress,event.destinationPort,  
event.deviceInboundInterface
```

However, you may have noticed that the type of each token was not defined; nor were any possible token formats. Because you can have several of these sub-messages for each file, the sub-message engine tries to deduce the type based on the mapping. This may not always work, so there is a way to explicitly set the types and the formats. Internally, the sub-message engine labels each token by its position in the regular expression (like Perl). In the engine, the tokens are named \$1, \$2, \$3, \$4, and so on, and you can set their type and format explicitly by defining the following properties:

```
submessage[0].pattern[0].types=String,IPAddress,Integer,  
IPAddress,Integer,String
```

The format can also be defined using one sub-message property (in this case, formats are not needed for the types specified. Use the keyword **null**):

```
submessage[0].pattern[0].formats=null,null,null,null,null,null
```

The combination of these last three properties:

```
submessage[0].pattern[0].fields=event.transportProtocol,  
event.sourceAddress,event.sourcePort,  
event.destinationAddress,event.destinationPort,  
event.deviceInboundInterface  
submessage[0].pattern[0].types=String,IPAddress,Integer,  
IPAddress,Integer,String  
submessage[0].pattern[0].formats=null,null,null,null,null,null
```

Will be internally equivalent to:

Six tokens

```
token.count=6
```

```
token[0].name=$1  
token[0].type=String  
token[0].format=null
```

```
token[1].name=$2  
token[1].type=IPAddress  
token[1].format=null
```

```
token[2].name=$3  
token[2].type=Integer
```



```
token[2].format=null

token[3].name=$4
token[3].type=IPAddress
token[3].format=null

token[4].name=$5
token[4].type=Integer
token[4].format=null

token[5].name=$6
token[5].type=String
token[5].format=null

event.transportProtocol=$1
event.sourceAddress=$2
event.sourcePort=$3
event.destinationAddress=$4
event.destinationPort=$5
event.deviceInboundInterface=$6
```

Using FlexConnector operations with the mapping properties of the sub-message is also possible. The following is an example with the sub-message for message ID 500004. The definition of that sub-message is as follows:

```
submessage[1].messageid=500004
submessage[1].pattern.count=1
submessage[1].pattern[0].regex=Invalid transport field for protocol\=(\\d+),
from (\\d+\\.\\d+\\.\\d+\\.\\d+)/ (\\d+) to (\\d+\\.\\d+\\.\\d+\\.\\d+)/ (\\d+)
submessage[1].pattern
[0].fields=event.applicationProtocol,event.sourceAddress,event.sourcePort,even
t.destinationAddress,event.destinationPort
```

Recall the original message:

```
Nov 29 01:35:15 10.0.111.2 %PIX-4-500004: Invalid transport field for
protocol=6, from 2.2.2.2/0 to 3.3.3.3/0
```

Notice that the `event.applicationProtocol` is mapped to \$1 which has the value `6`. The FlexConnector operation `__getProtocolName` translates protocol numbers into their description (for example, protocol number `6` is `TCP`). To use this operation, define a custom mappings property, so instead of `event.applicationProtocol=$1` use `event.applicationProtocol=__getProcotolName($1)`. Use the following property:

```
submessage[1].pattern[0].mappings=__getProtocolName($1)|$2|$3|$4|$5
```

In this case, each of the mappings is separated by a pipe (|) instead of a comma (,) because some operations could contain a comma. You can customize the delimiter if needed by setting the property `submessage[1].pattern[0].mappings.delimiter`. For example:

```
submessage[1].pattern[0].mappings.delimiter=@
submessage[1].pattern[0].mappings=
__getProtocolName($1)@$2@$3@$4@$5
```

Moving on to message ID 305005, notice that the same message ID has two slightly different formats. As mentioned before, sub-messages also support multiple regular expressions for a single message ID. The expressions are evaluated in order and the first match that succeeds wins. Try to order your expressions from the most specific to the most generic. The sub-message properties for message 305005 are as follows:

```
submessage[2].messageid=305005
submessage[2].pattern.count=2
submessage[2].pattern[0].regex=No translation group found for (\\S+) src
inside\\:(\\d+\\.\\d+\\.\\d+\\.\\d+)/ (\\d+) dst outside\\:
(\\d+\\.\\d+\\.\\d+\\.\\d+)/ (\\d+)
submessage[2].pattern[0].fields=event.transportProtocol,
event.sourceAddress,event.sourcePort,
event.destinationAddress,event.destinationPort
submessage[2].pattern[1].regex=Translation built for gaddr
(\\d+\\.\\d+\\.\\d+\\.\\d+) to laddr (\\d+\\.\\d+\\.\\d+\\.\\d+)
submessage[2].pattern[1].fields=
event.destinationTranslatedAddress,event.destinationAddress
```

Default Sub-message

There is one more sub-message feature that can be useful situations where you do not know every single message ID that can be received, but still want to try to parse them. In this case, define a default sub-message to use for any message with a message ID that is not defined (anything other than 106015, 305005, 500004, in this example, will be sent to the default sub-message). The default sub-message can also contain multiple patterns so that you can use several regular expressions to see if one of them matches.

The default sub-message is the same as a normal sub-message with no `messageid` property. The definition of a default sub-message for the current example configuration file will be:

```
submessage[3].pattern.count=1
submessage[3].pattern[0].regex=(.*)
submessage[3].pattern[0].fields=event.message
```

Of course, since new sub-message is added (the default sub-message is still a sub-message), increase the `submessage.count` to 4:

```
submessage.count=4
```

The default sub-message defined here will simply map the **event.message** to the full sub-message. You might want to alert the user that the particular message was not fully parsed; to do that, you can set the **event.name** to a fixed string, such as **Unparsed message** and the **deviceProduct** to **Unknown** so that separate statistics are kept for all these messages. See ["Extra Mappings"](#) for details.

Extra Mappings

Extra mappings (**extramappings**) is another property of the sub-message that can be used to directly add additional mapping properties. For the example described above, the **extramappings** property must be defined as:

```
submessage[3].pattern[0].extramappings=event.name=
__stringConstant("Unparsed event")
|event.deviceProduct=__stringConstant("Unknown")
```

Notice that you can add as many mappings as you require; each separated by '|'. The '|' can also be replaced with a different delimiter (just like the mappings delimiter):

```
submessage[3].pattern[0].extramappings.delimiter=@
submessage[3].pattern[0].extramappings=event.name=
__stringConstant("Unparsed event")
@event.deviceProduct=__stringConstant("Unknown")
```

Now the example FlexConnector with sub-messages is complete. The full FlexConnector configuration file looks like this:

FlexConnector Regex Configuration File

```
regex=(\\S+ \\d+ \\d+\\:\\d+\\:\\d+) (\\S+) %PIX-(\\d+)-(\\d+)\\: (.*)
```

```
token.count=5
```

```
token[0].name=Timestamp
token[0].type=TimeStamp
token[0].format=MMM dd HH\\:mm\\:ss
```

```
token[1].name=PixIP
token[1].type=IPAddress
```

```
token[2].name=PixSeverity
token[2].type=String
```

```
token[3].name=SubmessageIDToken
token[3].type=String
```

```
token[4].name=SubmessageToken
token[4].type=String

submessage.messageid.token=SubmessageIdToken
submessage.token=SubmessageToken

event.deviceReceiptTime=__useCurrentYear(Timestamp)
event.deviceAddress=PixIP
event.message=SubmessageToken
event.deviceVendor=__stringConstant(CISCO)
event.deviceSeverity=PixSeverity
event.deviceProduct=__stringConstant(PIX)
event.deviceEventClassId=SubmessageIDToken

severity.map.veryhigh.if.deviceSeverity=0,1
severity.map.high.if.deviceSeverity=2,3
severity.map.medium.if.deviceSeverity=4,5
severity.map.low.if.deviceSeverity=6,7

submessage.count=4

submessage[0].messageid=106015
submessage[0].pattern.count=1
submessage[0].pattern[0].regex=Deny (\\S+) \\(no connection\\) from
(\\d+\\.\\.\\d+\\.\\.\\d+\\.\\.\\d+)/ (\\d+) to (\\d+\\.\\.\\d+\\.\\.\\d+\\.\\.\\d+)/ (\\d+)
flags RST on interface (\\S+)
submessage[0].pattern[0].fields=event.transportProtocol,
event.sourceAddress,event.sourcePort,
event.destinationAddress,event.destinationPort,
event.deviceInboundInterface
submessage[0].pattern[0].types=String,IPAddress,Integer,
IPAddress,Integer,String
submessage[0].pattern[0].formats=null,null,null,null,null,null
submessage[1].messageid=500004
submessage[1].pattern.count=1
submessage[1].pattern[0].regex=Invalid transport field for protocol\\=(\\d+),
from (\\d+\\.\\.\\d+\\.\\.\\d+\\.\\.\\d+)/ (\\d+) to (\\d+\\.\\.\\d+\\.\\.\\d+\\.\\.\\d+)/ (\\d+)
submessage[1].pattern[0].mappings.delimiter=@

submessage[1].pattern[0].fields=event.applicationProtocol,
event.sourceAddress,event.sourcePort,
event.destinationAddress,event.destinationPort
submessage[1].pattern[0].mappings=
```

```
__getProtocolName($1)@$2@$3@$4@$5
submessage[2].messageid=305005
submessage[2].pattern.count=2
submessage[2].pattern[0].regex=No translation group found for (\\S+) src
inside\\:(\\d+\\.\\d+\\.\\d+\\.\\d+)/((\\d+) dst outside\\:
(\\d+\\.\\d+\\.\\d+\\.\\d+)/((\\d+)
submessage[2].pattern[0].fields=event.transportProtocol,
event.sourceAddress,event.sourcePort,
event.destinationAddress,event.destinationPort
submessage[2].pattern[1].regex=Translation built for gaddr
(\\d+\\.\\d+\\.\\d+\\.\\d+) to laddr (\\d+\\.\\d+\\.\\d+\\.\\d+)
submessage[2].pattern[1].fields=
event.destinationTranslatedAddress,event.destinationAddress
# Default sub-message descriptor
submessage[3].pattern.count=1
submessage[3].pattern[0].regex=(.*)
submessage[3].pattern[0].extramappings.delimiter=@
submessage[3].pattern[0].fields=event.message
submessage[3].pattern[0].extramappings=
event.name\\=__stringConstant("Unparsed event")
@event.deviceProduct\\=__stringConstant("Unknown")
```

Conditional Mappings

Conditional mappings enable you to map tokens that can contain different types of information, based on the characteristic of the event.

For example, assume the following event:

```
Event id is 532 type A with parameter 3.3.3.3
Event id is 533 type A with parameter root
Event id is 534 type A with parameter 3.3.3.3
```

In this example, the parameter token can be either an IP address or a user name.

The regular expression to parse this event is:

```
Event id is (\\d+) type (\\S+) with parameter (\\S+)
```

You can define three tokens for the above events: **EVENTID**, **TYPE**, and **PARAMETER**. If the event id is 532 or 534, set the ArcSight event field **event.sourceAddress** to 3.3.3.3 and if the event id is 533, set the **event.sourceUserName** to root.

Without conditional mappings, you will have to create two regular expressions to match the two unique information types in this event—the IP address and the user name. Although it is feasible to define two

regular expressions for this case, if there were hundreds of messages with unique information types, this solution will not scale well.

With conditional mappings, you can define the following mapping properties in your parser for the above example:

```
regex=Event id is (\\d+) type (\\S+) with parameter (\\S+)
token.count=3
token[0].name=EVENTID
token[1].name=TYPE
token[2].name=PARAMETER
#Standard mappings
event.deviceEventClassId=EVENTID
event.deviceEventCategory=TYPE
#Conditional mappings
conditionalmap.count=1
conditionalmap[0].field=event.deviceEventClassId
conditionalmap[0].mappings.count=2
conditionalmap[0].mappings[0].values=532,534
conditionalmap[0].mappings[0].event.sourceAddress=PARAMETER
conditionalmap[0].mappings[1].values=533
conditionalmap[0].mappings[1].event.sourceUserName=PARAMETER
```

The properties in the Conditional mappings section above define the following logic:

- **conditionalmap.count**—Specifies the number of conditional mappings. In the above example, one conditional mapping is defined.
- **conditionalmap[x].field** or **conditionalmap[x].token**—Specifies the field or token to evaluate. You can only use one of these properties for each conditional mapping, and not both.

When using **conditionalmap[x].field**, you must use the **event.eventIdField** format to specify a value for this property. In the above example, **conditionalmap[0].field=event.deviceEventClassId**.

When using **conditionalmap[x].token**, you must specify the token as the value. For example, **conditionalmap[0].token=PARAMETER** (not shown in the above example).

- **conditionalmap[x].mappings.count**—Specifies the count of information types. In the above example, 2—**sourceAddress** and **sourceUserName**.
- **conditionalmap[x].mappings[x].values**—Specifies a list of values to match with each token or field defined. In the example above, **conditionalmap[0].mappings[0].values = 532, 534**.

If you have more than one value, use a comma to separate them.

If this property is omitted, the conditional mapping is processed as a DEFAULT mapping that is executed ONLY if the previous mappings did not match. This is analogous to the **Else** behavior in the **If...Else** construct. For example, if the following conditional mapping was defined in addition to the mappings in the above example:

```
conditionalmap[0].mappings[2].event.destinationAddress=PARAMETER
```

Then, if an event with an event id other than 532, 533, and 534 was received, its **event.destinationAddress** will be set to **PARAMETER**. If you added the **DEFAULT** conditional map as suggested above to the previous example, then you must change the **conditionalmap[0].mappings.count** to 3 for the example to work.

- **conditionalmap[x].mappings[x].event.{xxxx}** or **conditionalmap[x].mappings[x].additionaldata.{xxx}**—Specifies the mapping properties to be evaluated if **conditionalmap[x].mappings[x].values** match the **conditionalmap[x].field** or **conditionalmap[x].token**.
- **conditionalmap[x].mappings[x].delimiter**—Specifies the delimiter to use for the values defined above. By default, comma (,). This property is optional.

Using Conditional Mapping in Sub-messages

You can use conditional mappings in sub-messages. For example:

```
submessage[3].messageid=conditionalmapsample
submessage[3].pattern.count=1
submessage[3].pattern[0].regex=Event id is (\\d+) type (\\S+) with parameter (\\S+)
submessage[3].pattern[0].fields=event.deviceEventClassId
submessage[3].pattern[0].conditionalmap.count=2
submessage[3].pattern[0].conditionalmap[0].field=event.deviceEventClassId
submessage[3].pattern[0].conditionalmap[0].mappings.count=2
submessage[3].pattern[0].conditionalmap[0].mappings[0].values=532,534
submessage[3].pattern[0].conditionalmap[0].mappings
[0].event.destinationAddress=$3
submessage[3].pattern[0].conditionalmap[0].mappings[1].values=533
submessage[3].pattern[0].conditionalmap[0].mappings
[1].event.destinationUserName=$3
submessage[3].pattern[0].conditionalmap[1].token=$2
submessage[3].pattern[0].conditionalmap[1].mappings.count=1
submessage[3].pattern[0].conditionalmap[1].mappings[0].values=B
submessage[3].pattern[0].conditionalmap[1].mappings
[0].event.destinationAddress=$3
```

The regular expression is divided into groups. A group is an element between two parentheses (). Each group is represented by **\$number** from left to right, where **number** is a sequentially increasing whole number, starting at 1.

In the above example, there are three groups:

```
$1 -- (\\d+)
```

\$2 -- (\\S+)

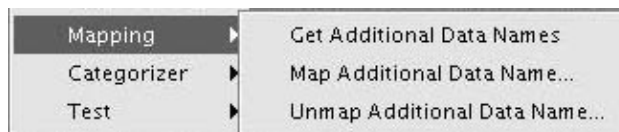
\$3 -- (\\S+)

Additional Data Mapping

In some environments it is useful to map certain additional data names to normal ArcSight schema fields. The mapping can vary based on the device vendor and product and can be controlled from the ArcSight Console, with the mappings stored on the SmartConnector machine.

The SmartConnector tracks whatever additional data names it encounters and reports this information to the ArcSight Console (otherwise, spelling and capitalization errors would make the mapping feature much more difficult to use.)

All data mapping is done through SmartConnector commands from the ArcSight Console, as shown:



Using the Get Additional Data Names Command

The **Get Additional Data Names** command specifies the additional data names assigned to each device vendor or product combination since the SmartConnector started running. This process has a default limitation of the most recent 100 device vendor/product combinations, and the most recent 100 names for each (this limit can be changed with the SmartConnector property **agent.additionaldata.mapper.track.max.names**).

The command output looks like this:

Additional Data Names Seen:

Generic (no vendor/product):

```
test1 [3 times]
test11
test13 [2 times]
test14 [3 times]
test15 [4 times]
test17 [5 times]
test18 [6 times]
test2 [4 times]
test20 [2 times]
test3 [5 times]
test4
test5 [3 times]
```

Vendor/product [vend/prod]:


```
test1
test10 [6 times]
test11
test12 [4 times]
test13 [2 times]
test14
test15 [2 times]
test17 [4 times]
test19 [2 times]
test2 [3 times]
test20 [4 times]
test5 [4 times]
test9
Vendor/product [vend/prod2]:
test10 [2 times]
test11 [5 times]
test12 [5 times]
test13 [7 times]
test15 [4 times]
test17 [2 times]
test18 [5 times]
test19
test2 [4 times]
test20 [6 times]
test3 [3 times]
test4 [6 times]
test6
test7
test9 [4 times]
```

If an additional data name appears more than once, the number of times it has been seen is included in the command output, as shown above.

Using the Map Additional Data Name... Command

The **Map Additional Data Name...** command opens this dialog:

Name	Value
Device vendor	
Device product	
Additional data name	
ArcSight field	

OK Cancel

The **Device vendor** and **Device product** fields can be left blank to create a generic mapping, or filled in for a specific mapping. The additional data name is usually one of the names shown in the **Get Additional Data Names** output, but does not have to be. The **ArcSight field** must be a valid ArcSight event field. The command output for a successful generic mapping looks as follows:

```
Successfully mapped additional data name [test11] to event field [message]
for vendor/product []
```

A successful device vendor/product-specific mapping has output similar to the following:

```
Successfully mapped additional data name [test10] to event field [message]
for vendor/product [vend/prod]
```

If the additional data name has not been seen, the name is still mapped, but with a warning as follows:

```
Successfully mapped additional data name [foo] to event field
[deviceCustomString1] for vendor/product [vend/prod] (note that additional
data name [foo] has not been seen for vendor/product [vend/prod])
```

If the **ArcSight field** is not valid, an error similar to the following is displayed:

```
Failed to map additional data name [bar] to event field [messages] for
vendor/product [vend/prod] (event field [messages] is unknown)
```

Using the Unmap Additional Data Name... Command

The **Unmap Additional Data Name...** command opens this dialog:

Name	Value
Device vendor	
Device product	
Additional data name	

OK Cancel

The **Device vendor** field and **Device product** fields can be left blank to remove a generic mapping, or be filled to remove a specific mapping. The additional data name should be one that was previously mapped for the specified device vendor and product combination.

The command output for a successful generic unmapping displays as follows:

```
Successfully unmapped additional data name [test11] for vendor/product []
```

A successful device vendor/product-specific unmapping has output similar to the following:

```
Successfully unmapped additional data name [foo] for vendor/product
[vend/prod]
```

If the specified additional data name was not previously mapped, the output displays as follows:

**Failed to unmap additional data name [foo] for vendor/product [vend/prod]
(not previously mapped)**

Note: One additional data name can be mapped to more than one **ArcSight field** for the same device vendor/product combination. In such cases, unmapping it unmaps it from all **ArcSight fields** for that device vendor/product.

In the opposite case, where multiple additional data names are mapped to the same **ArcSight field** for the same device vendor/product combination, the last mapping takes precedence over previous mappings to that **ArcSight field** and its corresponding device vendor/product combination.

Using the Get Status Command

The **Get Status** command includes the status for additional data names which are mapped to **ArcSight fields**, as shown below:

```
NGCustomAdditionalDataMapper0.....Generic mappings:test11=>message
NGCustomAdditionalDataMapper1.....Mappings for
vend/prod:test10=>message, foo=>deviceCustomString1
```

Note: Only mappings for loaded device vendor/product combinations are included. This includes mappings for vendor/product combinations that have had mapping or unmapping commands executed (even unsuccessful ones), and vendor/product combinations for which additional data-laden events have been seen. Unloaded mappings on disk are not included.

Log Rotation Types

For connectors that follow log files, there are three mechanisms for rotating the log files implemented in the connector framework. See "[File Connector Parameters](#)" for more information and the parameters available for log rotation.

- [Name Following Log Rotation](#)
- [Daily Rotation](#)
- [Index Rotation](#)

Several of the parameters are described here:

- [Parameters for Daily and Index Rotation](#)

Name Following Log Rotation

An example of name following log rotation would be, the device writes to **xyz.log**. At rotation time, the device renames **xyz.log** to **xyz1.log** and creates a new **xyz.log** and begins to write to it. The

connector detects the drop in size of **xyz.log** and terminates the reader thread to the old **xyz.log** after processing is completed. The connector creates a new reader thread to the new **xyz.log** and begins processing that file. To enable this log rotation, set **followexternalrotation** and **filesizecheck** to true.

Daily Rotation

A typical scenario of daily rotation could be, the device writes to **xyz.timestamp.log** on a daily basis. At a specified time, the device creates a new daily log and begins to write to it. The connector detects the new log and terminates the reader thread to the previous log after processing is complete. The connector then creates a new reader thread to the new **xyz.timestamp.log** and begins processing that file. To enable this log rotation, set **rotationscheme** to Daily. See also "[Parameters for Daily and Index Rotation](#)".

Index Rotation

In the case of index rotation, the device writes to indexed files - **xyz.log.001**, **xyz.log.002**, **xyz.log.003** and so on. At startup, the connector processes the log with highest index. When the device creates a log with a greater index, the connector terminates the reader thread to the previous log after processing completes, creates a thread to the new log and begins processing that log. To enable this log rotation, set **rotationscheme** to Index. See also "[Parameters for Daily and Index Rotation](#)".

Parameters for Daily and Index Rotation

Use the **rotationschemeparams** parameter to set the parameters for daily or index log file rotation. The **rotationschemeparams** parameter can be used only if the **rotationscheme** parameter is set to **Daily** or **Index**.

Using rotationschemeparams for Daily Log File Rotation

This section describes values for the **rotationschemeparams** parameter when **rotationscheme=Daily**. Applications use this value to generate date coded log files (for example, Trend Micro ScanMail).

A filename template has the following syntax:

[prefix,]dateFormat,suffix[,true|false]

Symbols used in the **dateFormat** can be read by the connector. They do not need to be declared as strings. For example:

yyyy.MM.dd
yyyy-MM-dd

Any character can be used in the **prefix** and **suffix** fields except the comma (,).

To include a literal string in a field, escape it with a single quote ('). For example:

```
access_,yyyyMMdd'in TimeZone: PST',.log,true
```

The [**prefix**] and [**true|false**] fields are optional. The [**true|false**] field indicates that the rotated file has an additional counter so it can be rotated multiple times a day. For example, to obtain the following output:

```
Access.yyyyMMdd.log.1
Access.yyyyMMdd.log.2
...
```

the syntax would have to be:

```
Access.,yyyyMMdd,.log,true
```

If you use periods (or “full stops”) within filenames, then they must be stated within the parameters. The commas which separate **prefix**,**dateFormat**,**appendix** do not replace them. For example, to obtain this output:

```
Filename.yyyyMMdd.appendix
```

use this syntax:

```
Filename.,yyyyMMdd,.appendix
```

Example:

```
yyyyMMdd,log
```

In this example, **prefix** is omitted, therefore, the following comma is not required. **dateFormat** is **yyyyMMdd**, **suffix** is **log**, and the [**true|false**] field is omitted. Because [**true|false**] is the last field and is omitted, a comma is not required at the end.

Example:

```
Access,yyyyMMdd,log
```

In this example, **prefix** is **Access**, **dateFormat** is **yyyyMMdd**, **suffix** is **log**, and the [**true|false**] field is omitted.

Example:

```
Access.,yyyyMMdd,
```

In this example the prefix is **Access.** and **yyyyMMdd** is the mandatory **dateFormat** field. The file does not have an suffix, but the configuration still must end with a comma to indicate that it is the end of the file name. This syntax will produce an output such as **Access.20160209**.

Example:

```
Access,yyyyMMdd,log,true
```

In this example, **prefix** is **Access**, **dateFormat** is **yyyyMMdd**, **suffix** is **log**, and the **[true|false]** field is set to **true**. Here **true** means even if the file name does not exactly match with the above given format, if the file name ends with the **suffix** and starts with **prefix** and also has the date in it then that file is matched.

For example: **Access_v2.20150225.log**

Using rotationschemeparams for Index Log File Rotation

This section describes values for the **rotationschemeparams** parameter when **rotationsscheme=Index**. For example:

my. '%03d,001,999,false'.log

The value **%03d** specifies how many digits are allowed before **.log** in the file name. In this example, 3 digits are allowed. The value **001,999** specifies how high to count in the index. In this example, the file rotation could go to **my.999.log**.

The last parameter, **[true|false]**, is optional. The default is **false**, which means missing indexes are not allowed. The connector does not stop reading the current file until the log file with the next index appears.

When **true**, it specifies that the connector continues processing if there is a missing file, for example if the device rotates the log from **my.636.log** to **my.638.log**.

Using wildcard for Daily and Index Log File Rotation (File Folder Follower Only)

Use the **agents[x].wildcard** parameter to match file names when rotating log files.

Note: The Regex File connector processes only files with the specified file extension.

To process all files for Regex File connectors on the Windows platform, use the value "asterisk dot asterisk" (***.***). because all files have an extension by default.

To process all files for Regex File connectors on the Unix/Linux platforms, the recommended value is "asterisk" (*****). For example, if you configure this property to ***.ext**, then the Regex connector will read events from only log files with the extension **.ext**.

Using wildcard for Date Rotation

A typical scenario could be, the device writes to **xyz.timestamp.log** on a daily basis. At a specified time, the device creates a new daily log and begins to write to it. The connector detects the new log and terminates the reader thread to the previous log after processing is complete. The connector then creates a new reader thread to the new **xyz.timestamp.log** and begins processing that file. To enable this log rotation, set **wildcard** to a data file format, as shown in the following example:

```
agents[x].wildcard=fileName.'yyyy-MM-dd'.fileSuffix
```

For a data file name of **myFile.2013-09-23.log**, the **wildcard** command is:

```
agents[x].wildcard=foo.'yyyy-MM-dd'.log
```

Where **myFile** is the **fileName**, **'yyyy-mm-dd'** is the date format, and **.log** is the **fileSuffix**.

Pattern matching is performed only for the portion within quotes.

Using wildcard for Index Rotation

In this case, the device writes to indexed files, for example: **xyz.001.log**, **xyz.002.log**, **xyz.003.log**, and so on. At startup, the connector processes the log with highest index. When the device creates a log with a greater index, the connector terminates the reader thread to the previous log after processing completes, creates a thread to the new log and begins processing that log. To enable this log rotation, set **wildcard** using the syntax shown in the following example:

```
agents
[x].wildcard=FileName.'patternOfIndex,minValue,maxValue,ignoreMissingIndex'.fileSuffix
```

where:

- **patternOfIndex** is the pattern of the index. It specifies how many digits are allowed before **.log** in the file name. For example, to allow a 3-digit index, enter **%d03**.
- **minValue** is the minimum value the index can take.
- **maxValue** is the maximum value the index can take, after which it again starts from **minValue**. For example, assume that **minValue**=000, and **maxValue**=999. When the connector finds a file with a 999 index, it will then look for a rotated file with the index 000.
- **ignoreMissingIndex** is a Boolean value that describes what the connector should do if a file with the next index is never created. If true, the connector checks if there is a new file with the correct file name pattern instead of waiting for the next index-based file forever.

For example, the command:

```
agents[x].wildcard=myFile.'%d01,0,9,true'.log
```

will support the processing of log files **myFile.0.log** through **myfile.9.log** before searching for **myFile.0.log** again.

Log Internal Events for File-Reading FlexConnectors

File-reading FlexConnectors have internal events that are sent when the connector begins to process a file and when the connector finishes processing the file. Another event can be configured so that the event is sent when a specified number of files are not processed in a specified amount of time. The events are configured in **ARCSIGHT_HOME/user/agent/agent.properties**.

- **internalevent.filestart.enable=true/false** - the default is true.
- **internalevent.fileend.enable=true/false** - the default is true.
- **internalevent.filecount.enable=false/true** - the default is false. This event has the following parameters:
 - **internalevent.filecount.duration=nnn** - specifies the number of seconds that the connector has to process a specified number of files.
 - **internalevent.filecount.minfilecount=nnn** - specifies the minimum number of files that the connector should process in a specified number of seconds.
 - **internalevent.filecount.timer.delay=nnn** - specifies, in seconds, how often the connector should check to see if the connector is compliant with the other parameters.

Unparsed Events Detection

The unparsed event detection feature syntactically detects unparsed events and logs them to a separate file for easier identification. This feature can be enabled by configuring the **unparsedevents.log.enabled** parameter and setting it to **true**. See "[Parameters Common to all SmartConnectors](#)" for more information about this parameter.

To verify whether the feature is enabled, see the **agent.log** file. The following sample log message indicates that the feature is enabled:

```
[2016-05-10 18:00:40,190][INFO ]  
[default.com.arcsight.agent.loadable.agent._DHCPFileAgent]  
[parseParameters] Logging of unparsed events is [enabled] for SmartConnector  
[dhcp_file][3vEFHnVQBABCAA9NWREbq5g==]
```

Supported Parser Types

Connectors with the following parser types can use the unparsed event detection feature:

- Regex parser—Configuration files for this parser type have the extension **sdkrfilereader.properties**.
- Key-Value parser—Configuration files for this parser type have the extension **sdkeyvaluefilereader.properties**.
- Delimited parser—Configuration files for this parser type have the extension **sdkfilereader.properties**.

To determine whether a connector uses any of these parser types, see the **agent.log** file. The following sample log message indicates that the connector uses a delimited parser:

```
[2016-05-10 18:00:40,222][INFO ]  
[default.com.arcsight.agent.content.FCPContentInputStreamProvider]
```



```
[getInputStream] Resource [dhcp_file\dhcp_file_v6.sdkfilereader.properties]  
found in [Z:\system\agent\fcg\arcsightagents.aup|dhcp_file\dhcp_file_  
v6.sdkfilereader.properties.arc]
```

Unparsed Events Detection Criteria

An ArcSight parser configuration file can contain any of these expressions:

- [Comment Expressions](#)
- [Parsing Expressions](#)
- [Token Expressions](#)
- [Mapping Expressions](#)
- [Extra-Processor Expressions](#)

To understand these expressions, consider the following parser file:

```
#  
# Parser file for Microsoft Windows DHCP File Agent  
#Event ID  Meaning  
#00        The log was started.  
#01        The log was stopped.  
#02        The log was temporarily paused due to low disk space.  
#10        A new IP address was leased to a client.  
#11        A lease was renewed by a client.  
#12        A lease was released by a client.  
#13        An IP address was found to be in use on the network.  
#14        A lease request could not be satisfied because the scope's  
#           address pool was exhausted.  
#15        A lease was denied.  
#16        A lease was deleted.  
#17        A lease was expired.  
#20        A BOOTP address was leased to a client.  
#21        A dynamic BOOTP address was leased to a client.  
#22        A BOOTP request could not be satisfied because the scope's  
#           address pool for BOOTP was exhausted.  
#23        A BOOTP IP address was deleted after checking to see it was  
#           not in use.  
#24        IP address cleanup operation has begun.  
#25        IP address cleanup statistics.  
#30        DNS update request to the named DNS server  
#31        DNS update failed  
#32        DNS update successful  
#50+       Codes above 50 are used for Rogue Server Detection information.
```

```
#DHCP 2008 QResult: 0: NoQuarantine, 1:Quarantine,  
2:Drop Packet, 3:Probation,6:No Quarantine Information ProbationTime:Year-  
Month-Day Hour:Minute:Second:MilliSecond.  
#ID,Date,Time,Description,IP Address,Host Name,MAC Address,User Name,  
TransactionID, QResult,Probationtime, CorrelationID,Dhcid.  
#DHCP 2003 ID,Date,Time,Description,IP Address,Host Name,MAC Address
```

```
regex=(\\d+),(\\d+\\/\\d+\\/\\d+),(\\d+:\\d+:\\d+),((?:?:.*?, )*)?.*?),(.*?),  
([ ^,]*),([-\\+]?\\w*),?  
line.ignore.regex=\\s*  
comments.start.with=#
```

```
token.count=7  
token[0].name=EventID  
token[0].type=String  
token[1].name=Date  
token[1].type=Date  
token[1].format=MM/dd/yy  
token[2].name=Time  
token[2].type=Time  
token[2].format=HH:mm:ss  
token[3].name=EventName  
token[3].type=String  
token[4].name=Address  
token[4].type=String  
token[5].name=HostName  
token[5].type=String  
token[6].name=sourceMAC  
token[6].type=String
```

```
event.sourceHostName=HostName  
event.deviceEventClassId=EventID  
event.name=EventName  
event.deviceReceiptTime=__createTimeStamp(Date,Time)  
#Convert address for event id = 30 - DNS update request  
event.sourceAddress=__splitAsAddress(__ifThenElse(EventID,"30",  
__reverseDottedDecimalAddressByteOrder(Address),),,  
event.deviceProduct=__stringConstant("DHCP Server")  
event.deviceVendor=__getVendor("Microsoft")  
event.deviceCustomString4=__toUpperCase(__regexTokenNoWarning  
(sourceMAC,"(\\S{1,6}).*"))
```

```
event.sourceMacAddress=__getLongMACAddressByHexString(sourceMAC)
event.deviceCustomString4Label=__stringConstant(MAC Vendor Prefix)
event.deviceCustomString5Label=__stringConstant(Ethernet Vendor)
event.deviceCustomNumber1=__safeToLong(__ifThenElse(EventID,"25",
__regexTokenNoWarning(EventName,"(\\d+) leases.*"),))
event.deviceCustomNumber2=__safeToLong(__ifThenElse(EventID,"25",
__regexTokenNoWarning(EventName,".* and (\\d+) leases.*"),))
event.deviceCustomNumber1Label=__ifThenElse(EventID,"25",
__stringConstant(leases expired),)
event.deviceCustomNumber2Label=__ifThenElse(EventID,"25",
__stringConstant(leases deleted),)
```

```
extraprocessor.count=2
extraprocessor[0].type=map
extraprocessor[0].filename=dhcp_file/event_ref.csv
extraprocessor[1].type=map
extraprocessor[1].filename=dhcp_file/ethernet_vendor_ref.csv
```

Comment Expressions

The following lines in the parser file represent the comment expressions:

```
#
# Parser file for Microsoft Windows DHCP File Agent
#Event ID  Meaning
#00        The log was started.
#01        The log was stopped.
#02        The log was temporarily paused due to low disk space.
#10        A new IP address was leased to a client.
#11        A lease was renewed by a client.
#12        A lease was released by a client.
#13        An IP address was found to be in use on the network.
#14        A lease request could not be satisfied because the scope's
#           address pool was exhausted.
#15        A lease was denied.
#16        A lease was deleted.
#17        A lease was expired.
#20        A BOOTP address was leased to a client.
#21        A dynamic BOOTP address was leased to a client.
#22        A BOOTP request could not be satisfied because the scope's
#           address pool for BOOTP was exhausted.
#23        A BOOTP IP address was deleted after checking to see it was
#           not in use.
```

```
#24      IP address cleanup operation has began.
#25      IP address cleanup statistics.
#30      DNS update request to the named DNS server
#31      DNS update failed
#32      DNS update successful
#50+     Codes above 50 are used for Rogue Server Detection information.
```

```
#DHCP 2008  QResult: 0: NoQuarantine, 1:Quarantine, 2:Drop Packet,
3:Probation,6:No Quarantine Information ProbationTime:Year-Month-Day
Hour:Minute:Second:MilliSecond.
#ID,Date,Time,Description,IP Address,Host Name,MAC Address,User Name,
TransactionID, QResult,Probationtime, CorrelationID,Dhcid.
#DHCP 2003 ID,Date,Time,Description,IP Address,Host Name,MAC Address
```

```
#Convert address for event id = 30 - DNS update request
```

Parsing Expressions

The following lines in the parser file indicate the parsing expressions. This parsing expression indicates how an event should be broken down and tokenized by the parser.

```
regex=(\\d+),(\\d+\\/\\d+\\/\\d+),(\\d+:\\d+:\\d+),((?:(?:.*?, )*)?.*?),(.*),
([^\,]*) ,([-\\+]?\\w*),?
line.ignore.regex=\\s*
comments.start.with=#
```

Token Expressions

The following lines in the parser file indicate the token expressions, that is, how many tokens to capture, the token name, token data type, and so on.

```
token.count=7
token[0].name=EventID
token[0].type=String
token[1].name=Date
token[1].type=Date
token[1].format=MM/dd/yy
token[2].name=Time
token[2].type=Time
token[2].format=HH:mm:ss
token[3].name=EventName
token[3].type=String
token[4].name=Address
```

```
token[4].type=String
token[5].name=HostName
token[5].type=String
token[6].name=sourceMAC
token[6].type=String
```

Mapping Expressions

The following lines in the parser file represent mapping expressions to indicate how the captured tokens should be mapped to ArcSight event schema fields.

```
event.sourceHostName=HostName
event.deviceEventClassId=EventID
event.name=EventName
event.deviceReceiptTime=__createTimeStamp(Date,Time)
event.sourceAddress=__splitAsAddress(__ifThenElse(EventID,"30",
__reverseDottedDecimalAddressByteOrder(Address),,))
event.deviceProduct=__stringConstant("DHCP Server")
event.deviceVendor=__getVendor("Microsoft")
event.deviceCustomString4=__toUpperCase(__regexTokenNoWarning
(sourceMAC,"(\\S{1,6}).*"))
event.sourceMacAddress=__getLongMACAddressByHexString(sourceMAC)
event.deviceCustomString4Label=__stringConstant(MAC Vendor Prefix)
event.deviceCustomString5Label=__stringConstant(Ethernet Vendor)
event.deviceCustomNumber1=__safeToLong(__ifThenElse(EventID,"25",
__regexTokenNoWarning(EventName,"(\\d+) leases.*"),))
event.deviceCustomNumber2=__safeToLong(__ifThenElse(EventID,"25",
__regexTokenNoWarning(EventName,".* and (\\d+) leases.*"),))
event.deviceCustomNumber1Label=__ifThenElse(EventID,"25",
__stringConstant(leases expired),)
event.deviceCustomNumber2Label=__ifThenElse(EventID,"25",
__stringConstant(leases deleted),)
```

Extra-Processor Expressions

The following lines in the parser file indicate the extra-processor expressions, to hand off the event to another parser file for further processing.

```
extraprocessor.count=2
extraprocessor[0].type=map
extraprocessor[0].filename=dhcp_file/event_ref.csv
extraprocessor[1].type=map
extraprocessor[1].filename=dhcp_file/ethernet_vendor_ref.csv
```

Criteria for Unparsed Events

If an event fails to tokenize based on the parsing expression used by the parser, then it is considered to be an unparsed event. The criteria for an event to be labeled an unparsed event is its failure to pass the parsing expression.

Note the following considerations on parsing criteria:

- Multi-line merging parsers, token operations, sub-messages, and conditional maps are out of scope of the detection criteria.
- Extra-processors that belong to the supported parser types are included in the detection criteria.

Example:

The following event line completely matches the parsing expression of the parser, hence it is considered to be a parsed event:

```
11000,03/23/15,12:43:35,DHCPV6 Solicit,2001:db8::f80f:9757:b0a5:c40c,2k12-  
dhcpsvr.fadetoblack.  
local,,14,000100011C87C704000C290FFAAF,,,,,
```

However, the following event line with an incorrect date string of **03/23** does not match the parsing expressions of the parser, hence it is considered to be an unparsed event:

```
11000,03/23,12:43:35,DHCPV6 Solicit,2001:db8::f80f:9757:b0a5:c40c,2k12-  
dhcpsvr.fadetoblack.  
local,,14,000100011C87C704000C290FFAAF,,,,,
```

This unparsed event also generates an exception stack trace in the **agent.log** file. The following is a sample stack trace:

```
[2016-03-10 18:00:41,031][ERROR]  
[default.com.arcsight.agent.dhcp.DhcpFileProcessor][processLine]  
[java.text.ParseException: Unparseable date: "03/23"  
    at java.text.DateFormat.parse(DateFormat.java:357)  
    at com.arcsight.agent.parsers.token.DateParser.parseToken  
(DateParser.java:105)  
    at com.arcsight.agent.sdk.parsers.SDKCustomParser.addToken  
(SDKCustomParser.java:292)  
    at com.arcsight.agent.dhcp.DhcpSemiConfigurableParser.parseTokens  
(DhcpSemiConfigurableParser.java:307)  
    at com.arcsight.agent.parsers.GenericParserImpl.parseValues  
(GenericParserImpl.java:397)  
    at com.arcsight.agent.parsers.GenericParserImpl.parse  
(GenericParserImpl.java:755)  
    at com.arcsight.agent.parsers.GenericParserImpl.parseString
```

```
(GenericParserImpl.java:806)
    at com.arcsight.agent.baseagents.filereader.multifile.FileProcessor.
parseLine(FileProcessor.java:202)
    at com.arcsight.agent.baseagents.filereader.multifile.FileProcessor.
processLine(FileProcessor.java:186)
    at com.arcsight.agent.baseagents.filereader.
NameFollowingFileReaderThread.processLine
(NameFollowingFileReaderThread.java:769)
    at com.arcsight.agent.baseagents.filereader.
BaseAutoConfigParserFileReaderThread.processLine
(BaseAutoConfigParserFileReaderThread.java:157)
    at com.arcsight.agent.dhcp.DhcpFileReaderThread.processLine
(DhcpFileReaderThread.java:79)
    at com.arcsight.agent.baseagents.filereader.FileReaderThread.run
(FileReaderThread.java:859)
    at java.lang.Thread.run(Thread.java:745)
```

When an event line fails to match the parsing expression of the parser, then it is considered to be an unparsed event. This information is logged in the **agent.log** file. The following is an example message:

```
[2016-03-10 18:00:41,027][ERROR]
[default.com.arcsight.common.log.EventLogManager]
[logUnparsedEvent] Cannot parse raw event [11000,03/23,12:43:35,DHCPV6
Solicit,2001:db8::f80f:9757:b0a5:c40c,2k12-dhcpv6.fadeto.black.
local,,14,000100011C87C704000C290FFAAAF,,,,] with ArcSight SmartConnector
[class com.arcsight.agent.loadable.agent._DHCPFileAgent], and Parser [class
com.arcsight.agent.dhcp.DhcpSemiConfigurableParser]. Parser Result: [].
Parsing Exception: [Unparseable date: "03/23"].
```

If an exception occurs when parsing the event, then it is also logged in the **agent.log** file. The following is an example exception message:

```
[2016-03-10 18:00:41,028][ERROR]
[default.com.arcsight.common.log.EventLogManager]
[logUnparsedEvent]
java.text.ParseException: Unparseable date: "03/23"
    at java.text.DateFormat.parse(DateFormat.java:357)
    at com.arcsight.agent.parsers.token.DateParser.parseToken(DateParser.
java:105)
    at com.arcsight.agent.sdk.parsers.SDKCustomParser.addToken
(SDKCustomParser.java:292)
    at com.arcsight.agent.dhcp.DhcpSemiConfigurableParser.parseTokens
(DhcpSemiConfigurableParser.java:307)
    at com.arcsight.agent.parsers.GenericParserImpl.parseValues
```

```
(GenericParserImpl.java:397)
    at com.arcsight.agent.parsers.GenericParserImpl.parse
(GenericParserImpl.java:755)
    at com.arcsight.agent.parsers.GenericParserImpl.parseString
(GenericParserImpl.java:806)
    at com.arcsight.agent.baseagents.filereader.multifile.FileProcessor.
parseLine(FileProcessor.java:202)
    at com.arcsight.agent.baseagents.filereader.multifile.FileProcessor.
processLine(FileProcessor.java:186)
    at com.arcsight.agent.baseagents.filereader.
NameFollowingFileReaderThread.processLine(
NameFollowingFileReaderThread.java:769)
    at com.arcsight.agent.baseagents.filereader.
BaseAutoConfigParserFileReaderThread.processLine
(BaseAutoConfigParserFileReaderThread.java:157)
    at com.arcsight.agent.dhcp.DhcpFileReaderThread.processLine
(DhcpFileReaderThread.java:79)
    at com.arcsight.agent.baseagents.filereader.FileReaderThread.run
(FileReaderThread.java:859)
    at java.lang.Thread.run(Thread.java:745)
```

Unparsed Events Output File

Unparsed events detected by the connector are logged to the **%ARCSIGHT_HOME%/logs/events.log** (Linux) or **\$ARCSIGHT_HOME/logs/events.log** (Windows) file. The following is a sample message:

```
"Timestamp","ArcSight SmartConnector","ArcSight Parser","Parser
Result","Parsing Exception","Unparsed Event"

"2016-05-10 18:00:41.030 -0700","class com.arcsight.agent.loadable.agent._
DHCPFileAgent","class com.arcsight.agent.dhcp.DhcpSemiConfigurableParser","",
"Unparseable date: ""03/23""", "11000,03/23,12:43:35,DHCPV6
Solicit,2001:db8::f80f:9757:b0a5:c40c,2k12-dhcpvr.fadetoblack.
local,,14,000100011C87C704000C290FFAAF,,,,,"
```

The **events.log** file is a CSV file containing the column headers on the first line and the unparsed events on the following lines. The following table describes the columns in the CSV file:

Column Name	Column Description	Sample Value	Required/Optional
Timestamp	The time stamp at which the event was detected as an unparsed event	2016-05-10 18:00:41.030 -0700	Required
ArcSight SmartConnector	The ArcSight SmartConnector class that detected the unparsed event class	com.arcsight.agent.loadable.agent._DHCPFileAgent	Required
ArcSight Parser	The ArcSight parser class that detected the unparsed event class	com.arcsight.agent.dhcp.DhcpSemiConfigurableParser	Required
Parser Result	The parser result, if any		Optional
Parsing Exception	The parser exception message, if any	Unparseable date: "03/23"	Optional
Unparsed Event	The unparsed event string	11000,03/23,12:43:35,DHCPV6 Solicit,2001:db8::f80f:9757:b0a5:c40c,2k12-dhcpsvr.fadetoblack.local,,14,000100011C87C704000C290FFAAF,,,,	Required

Chapter 7: Map Files

The following topics are covered in this chapter:

- [What Are Map Files?](#)
- [Map File Examples](#)
- [Map File Details](#)
- [Using Ranges in Map Files](#)
- [Using Regular Expressions in Map Files](#)
- [Using Parser-Like Expressions in Map Files](#)
- [Real World Examples](#)

What Are Map Files?

Map files are actual physical files, located in the connector itself. Map files operate on events after they are collected and parsed, but before they are sent to the destination, conditionally changing one or more event fields. There are several parts of the connector code that use map files:

- Basic map files, which operate on events early in the event flow
- **AgentInfoAdder1** map files, which operate on events later in the event flow, and can be made to operate differently when there are multiple destinations and/or multiple connectors running in one container
- The categorizer modules use map files to do their work
- Map file “extra processors” can be specified in FlexConnector parsers

Note: Map files are kept in memory for performance reasons, so large ones will affect the memory usage of the connector.

Map File Examples

A map file is a comma-separated file that you can edit in a text editor (such as Notepad or vi, which do not add formatting) or in a spreadsheet. The following is an example of a simple map file. In this document, map file examples are shown in tables for clarity.

Note: If you use a spreadsheet application to create or edit your map files, be sure to save the resulting files in the comma-separated value (CSV) format.

The first line normally defines the event fields that will be looked at ("getters") and those that will be potentially set ("setters"). Optionally, there can be a line before that, starting with **!Flags**, that controls certain values (see "[Controlling Map File Operation](#)"). In that case, it's the second line that defines the "getters" and "setters." A simple example without a **!Flags** line is:

event.destinationPort	set.event.applicationProtocol
20	ftp
21	ftp
80	http
110	pop3

In this example, the **applicationProtocol** event field is set based on the value of the **destinationPort** event field, but only if the **destinationPort** is one of the values in the "getter" column. If **destinationPort** is 21, **applicationProtocol** will be set to **ftp**, but if **destinationPort** is 22, **applicationProtocol** will not be set at all, because the value 22 does not appear in the **destinationPort** "getter" column.

There is a duplicate value (ftp) in the **applicationProtocol** column, which is allowed because it is a "setter", but not in the **destinationPort** column, in which a duplicate value would be an error.

This example would look like this in a text editor:

```
event.destinationPort,set.event.applicationProtocol
20,ftp
21,ftp
80,http
110,pop3
```

Multiple "Getters" and "Setters"

More complicated map files can have multiple "getter" columns (the row is only used if all column values match the event) and/or multiple "setter" columns (to set more than one field).

The following is an example with two "getters:"

event.deviceCustomNumber1	event.deviceEventCategory	set.event.deviceEventCategory
1	1	Vulnerability - Buffer/Heap Overflow
3	1	Vulnerability - Configuration Error
1	2	Malicious Code - Worm

In this case, if the **deviceCustomNumber1** and **deviceEventCategory** event fields are both 1, then the value for the **deviceEventCategory** event field is changed to **Vulnerability-Buffer/Heap Overflow**. If they are 3 and 1, respectively, the value is set to **Vulnerability-Configuration**

Error, and if the values are 1 and 2, the value is set to **Malicious Code-Worm**. Any other combination leaves the **deviceEventCategory** event field unchanged.

This example (and the next one) also shows that you can have a "getter" event field also appear as a "setter."

The following is an example with two "setters:"

event.name	set.event.name	set.event.deviceEventClassId
accept(2)	AUE_ACCEPT	AUE_ACCEPT
access(2)	AUE_ACCESS	AUE_ACCESS
acct(2)	AUE_ACCT	AUE_ACCT

In this case, the name event field is looked up to both replace the name event field and set the **deviceEventClassId** event field.

Also, you can have both multiple "getters" and multiple "setters" in the same map file.

Using the “No Getter” Trick

By having no “getters,” you can set one or more fields to specific constant values, unconditionally. For example:

set.event.message
Map file was here

This type of map file always has exactly two lines. It can have more than one column if you want to set more than one field, like this:

set.event.message	set.event.deviceCustomString1
Map file was here	And also here

Map File Details

The following subsections describe details for the various map file types.

Controlling Map File Operation

Any map file can be controlled with an optional initial line starting with **!Flags**, that can be omitted. If this line is present, it precedes the line that defines the "getters" and "setters." It is a comma-separated line similar to the rest of the file, but the number of columns do not have to match the other lines. The line must begin with **!Flags**, followed by one or more of the following flags, with commas in between:

Flag	Description
Overwrite	allow fields that are already set to be overwritten
Overwrite-	fields that are already set will not be overwritten
CaseSens	"getters" are case sensitive
CaseSens-	"getters" are case insensitive
TrimGetters	any leading or trailing whitespace or tabs are removed from "getters"
TrimGetters-	any leading or trailing whitespace or tabs are not removed from "getters"
TrimSetters	any leading or trailing whitespace or tabs are removed from "setters"
TrimSetters-	any leading or trailing whitespace or tabs are not removed from "setters" are
EnfrcUniqID	duplicate "getter" values are treated as fatal errors
EnfrcUniqID-	duplicate "getter" values are treated as warnings

Note: The minus sign after the flag reverses its meaning.

For example, the following would make "getters" case insensitive, not allow overwriting fields, and not remove any leading or trailing whitespace or tabs from "setters":

!Flags,CaseSens-,Overwrite-,TrimSetters-

Basic Map Files

Place basic map files in the **user/agent/map** directory under the ArcSight home directory of the connector file system. Name the files **map.0.properties**, **map.1.properties**, and so on. Basic map files are named as properties files (with the **.properties** extension), but they are actually CSV files.

New or changed map files will be automatically applied approximately every five minutes. Also, you can use the **Reload custom map files** command in the ArcSight Console to reload the basic map files on demand. See the *ArcSight Console User's Guide*, "Managing SmartConnector", "Sending Control Commands to SmartConnectors", under the "Categorizer" mapper category.

The files are numbered so that the connector knows what order to apply them, since changes made by one map file may affect a later map file. The numbering sequence must stay consecutive and files cannot be skipped. For example, the sequence 0, 1, 2, 3 is valid. The sequence 0, 1, 3 is not, and will cause the reading of the files to be stopped at 1 in this example.

By default, basic map files overwrite the values in event fields. Any leading or trailing whitespace or tabs are removed (trimmed) from "getters" and "setters", "getters" are case sensitive, and duplicate "getters" generate warnings. Any of these default behaviors can be changed with the **!Flags** line.

AgentInfoAdder1 Map Files

Put the files in the **user/agent/aup/acp** directory under the ArcSight home directory. Or use the **user/agent/aup/<id>/acp** directory for destination/connector-specific files, where **<id>** is replaced with the actual ID of the connector or destination. Name the files **AgentInfoAdder1.map.10.csv**, **AgentInfoAdder1.map.11.csv**, and so on.

The **AgentInfoAdder1** map files are numbered starting at 10, not 0 or 1, since files 0 to 9 are reserved for internal map files that are not visible to users. The files are numbered so that the connector knows what order to apply them, since changes made by one map file may affect a later map file. If there is a missing number (like files 10 and 12 but not file 11), no files after the missing number will be processed. Restart the connector to reload **AgentInfoAdder1** map files.

AgentInfoAdder1 map files will not overwrite event fields that are already set. By default, leading and trailing spaces are removed from “getter” and “setter” values before processing. “Getter” values are not case sensitive. If two rows have duplicate “getters”, a warning is logged. This is the default behavior of basic map files. These default behaviors can be modified by using the **!Flags** line.

Categorizer Map Files

Connectors categorize events, which is to say that the category fields (for example, **categorySignificance** and **categoryTechnique**) are set. The mechanism described here can categorize events that otherwise would not be categorized. And in fact that is key, because any event that has already been categorized will not be modified.

Put the files under the **user/agent/aup/acp/categorizer/current** directory under the ArcSight home directory. Under that, create a directory that matches the **deviceVendor** field of the events you want to categorize, and under that create a map file named for the **deviceProduct** field of the events you want to categorize, with the **.csv** extension. The **deviceVendor** and **deviceProduct** names must be modified as follows:

- Convert any uppercase letters to lowercase.
- Convert any characters that are not letters or digits to underscore characters.

For example, if the events will have **deviceVendor** set to "Giant Corp" and **deviceProduct** set to "It's a Big Product", then you would create **user/agent/aup/acp/categorizer/current/giant_corp/it_s_a_big_product.csv**.

This map file is just like any other map file, though they often only have one getter, on the **deviceEventClassId** field, and generally only set the category fields (**categoryObject**, **categoryBehavior**, **categoryTechnique**, **categoryDeviceGroup**, **categorySignificance**, and **categoryOutcome**).

You can use the **Reload custom categorizations** command in the ArcSight Console to reload the categorizer map files on demand. See the *ArcSight Console User's Guide*, "Managing SmartConnectors", "Sending Control Commands to SmartConnectors", under the "Categorizer" mapping category.

Categorizer map files can overwrite the values in event fields, though that rarely matters since events that have any of the category fields set will not be processed. Leading and trailing spaces are removed from "getter" and "setter" values before processing. The "getter" values are not case sensitive. If two rows have duplicate "getters", a warning is logged.

Extra Processor Map Files

See ["Extra Processors"](#) for general information on extra processors. An example of parser contents follows:

```
extraprocessor.count=1
extraprocessor[0].type=map
extraprocessor[0].filename=customvendor/customproduct.csv
extraprocessor[0].allowoverwrite=false
extraprocessor[0].casesensitive=false
extraprocessor[0].charencoding=US-ASCII
extraprocessor[0].trimgettertokens=false
extraprocessor[0].trimsettertokens=false
```

In this case the map file is the **user/agent/aup/fcp/customvendor/customproduct.csv** file. The other optional properties let you change the default operation, which allows overwriting values, is case sensitive, removes leading or trailing whitespace or tabs from "getters" and "setters", and uses the platform's default character encoding. The map file is just like any other map file, and operates on the event after the parser and any extra processors earlier in the list (if **extraprocessor.count** is greater than 1) is finished. If you need more than one map file, adjust **extraprocessor.count** accordingly and specify them.

Using Ranges in Map Files

You can use ranges in map files to simplify map file creation. For example, a map file that lists many source addresses can be quite large:

event.sourceAddress	set.event.deviceCustomString1
1.0.1.0	China
1.0.1.1	China
1.0.1.2	China

event.sourceAddress	set.event.deviceCustomString1
1.0.1.3	China
...763 more addresses...	
1.0.3.255	China

The above example would list 768 addresses, if the entire map file was shown.

Using a range in a map file, you can create a simpler file that does the same task. For example:

range.event.sourceAddress	set.event.deviceCustomString1
1.0.1.0-1.0.3.255	China

The resulting map file is easier to create, and is smaller and less prone to typing errors.

Ranges can be used on:

- Number event fields like **sourcePort** or **fileSize**
- IP address fields like **sourceAddress** and **deviceCustomIPv6Address1** (each range in the map file must be either IPv4 or IPv6, meaning it cannot have an IPv4 starting address and an IPv6 ending address, or vice versa. For IPv6-aware parsers, the map file should expect the possibility of either IP address type in any IP address field. For a non-IPv6-aware parser, the map file would only expect IPv4 in the normal fields and IPv6 in the **deviceCustomIPv6Address** fields)
- MAC address event fields like **destinationMacAddress**

Additional details pertaining to ranges:

- IPv6 addresses can use the :: and dotted-quad formats. In IPv6-aware parsers, IPv6 addresses can be used where they were previously not valid.
- MAC addresses must be in hexadecimal with colon separators
- Use the hyphen character as the separator between the lower bound and upper bound in the range
- Avoid overlapping ranges in the same column

Using Regular Expressions in Map Files

You can use regular expressions in map files to provide look up functionality to set field values. For example:

regex.event.sourceUserName	set.event.deviceCustomString1
.*?arcsight.com.*	ArcSight
.*?somesoft.com.*	Somesoft

In this example, the **sourceUserName** event field is looked up to see if it matches either of the regular expressions, and if it does, the **deviceCustomString1** event field is set accordingly.

The regular expression “getter” event field must be a string field, and the value in each event is matched against all of the regular expressions in that column. Unlike with ranges, it’s more difficult to avoid regular expressions that “overlap,” and the rule is that in that case the first one wins.

If you combine regular expressions with ranges, and there are no overlapping ranges (overlapping ranges are not recommended), it is best to put the ranges before (to the left of) the regular expressions, for performance reasons.

Using Parser-Like Expressions in Map Files

You can use parser-like expressions in map files to extend the functionality of map files.

Here is an example with three input events:

deviceCustomNumber1	deviceCustomString1	deviceCustomString3
1	“Leading and trailing”	“Whatever”
10	“Anyone reading this?”	“Overwrite with this”
17	“Hello”	“...there!”

These are the resulting **deviceCustomString1** values that we want for those three events:

deviceCustomString1
“Leading and trailing”
“Overwrite with this”
“Hello”

Unlike ranges and regular expressions, this feature isn’t about the “getters,” but about the “setter(s)”. In this example, we want to remove (trim) leading and trailing spaces from **deviceCustomString1** when the number is 1, and copy the value of **deviceCustomString3** into **deviceCustomString1** if the number is 10. For any other number, no change is desired.

Here is an example of a map file that can achieve the result shown above:

event.deviceCustomNumber1	set.expr (deviceCustomString1 deviceCustomString3).event.deviceCustomString1
1	__stringTrim(deviceCustomString1)
10	deviceCustomString3

Additional details:

- The “getter” column (or columns) controls which row, if any, is used
- In the header line, the expression “setter” lists what event fields might be used in the expressions in that column, inside the parentheses, and what event field will be set, at the end
- Then one of the actual expressions below that is evaluated and the result put into the event field

Note: Operations (such as `__stringTrim`) are described in ["ArcSight Operations"](#).

In this case **deviceCustomString1** and **deviceCustomString3** are listed inside the parentheses in the header row since they are used as described in ["More About Parser-Like Expressions Syntax"](#).

More About Parser-Like Expressions Syntax

For parser-like expressions, the “setter” header has several parts:

- Two constant parts: “set.expr(“ and “).event.”
- Between those is the list of event fields and/or additional data fields that might be used in the expressions, separated by pipes (two pipes separate event fields from additional data)
- The one event field that will be set to the result of the expression

Note: Expression “setters” cannot be used to set additional data fields, only event fields.

Below is a “no getter” example:

```
set.expr(deviceCustomNumber1|deviceCustomNumber2||addnumber).event.deviceCustomNumber3
```

```
"_sum(deviceCustomNumber1,deviceCustomNumber2,_safeToInteger(addnumber))"
```

This example sets **deviceCustomNumber3** to the sum of **deviceCustomNumber1**, **deviceCustomNumber2**, and (if it is a valid number) the additional data field **addnumber**.

Operations Containing Commas

When an operation contains any commas, most commonly with operations that have multiple arguments (for example, `__regexToken`), use quotes around the entire operation, and then change any quote characters that are now inside the outer quotes to two quote characters. The CSV parsing code will turn those doubled quote characters back into one quote character. For example:

```
"__regexToken(proto, "\".*?/(.*)\"")"
```

Backslashes in Expressions Versus in Parsers

In parsers you must use `\\` to represent one backslash character, but in these expressions you do not need to use the double backslash. Parsers are properties files, which use backslashes for quoting. Map files are CSV files (regardless what the file extension is), which use actual quotes for quoting.

Real World Examples

This section contains the following information:

- [Adding Country Names to Events](#)
- [Getting Domain Name from Hostname](#)

Adding Country Names to Events

In the following example, the goal is to add new fields to events that contain the name of the source and destination countries, based on the **sourceAddress** and **destinationAddress** event fields.

The data divides the IPv4 address space into many ranges, each of which is associated with a particular country. The map files are large enough (order of magnitude 100K lines) that you might need to increase the connector heap size. The resulting map file would look like this:

range.event.sourceAddress	set.additionaldata.SCN
1.0.0.0-1.0.0.255	Australia
1.0.1.0-1.0.3.255	China
1.0.4.0-1.0.7.255	Australia
... <i>additional lines in the map file</i> ...	

This example uses the range feature on an IPv4 event field. A second map file with the same data is also needed for the **destinationAddress** event field.

Getting Domain Name from Hostname

The map file example below uses the last two part of a hostname to get the domain name only.

set.expr(sourceHostName).event.deviceCustomString2
"__regexToken(sourceHostName, "***\\.([\\.]+\\.([\\.]+)\$\$\$")"

This table shows the results of this map file:

sourceHostName	deviceCustomString2
14-202-33-238.static.tpgi.com.au	com.au
bzq-79-181-26-177.red.bezeqint.net	bezeqint.net
dynamic-27-121-217-28.goi.ne.jp	ne.jp

soureHostName	deviceCustomString2
05405efe.skybroadband.com	skybroadband.com
dail-95-105-128-25-orange.orange.sk	orange.sk
host-19-157-66-217.spbmts.ru	spbmts.ru
118-160-227-230.dynamic.hinet.net	hinet.net

Appendix A: ArcSight Operations

The following table describes all of the operations that can be used when tokens are mapped to Micro Focus ArcSight event fields.

Operation	Return Type	Definition and Comments
__BASE64Decode	String	The parameter is a single Base-64 encoded string, which is decoded to bytes, and then converted to a string using the platform's default character set.
__byteArrayToIPAddress	IPAddress	This operation takes a byte array representation of an IPv4 or IPv6 address as a parameter and returns an IPAddress object. This operation can be used only for IPv6-aware parsers.
__byteArrayToIPv6	IPAddress	This parameter returns an IPv6 address stored as an IPAddress object. Use this parameter for mapping to event fields or additional fields which can have an IPv6 address type. Use this operation only in a non-IPv6-aware parser. For an IPv6-aware parser use the __byteArrayToIPAddress operation.
__byteArrayToIPv6String	String	The parameter returns the string representation of an IPv6 address stored in a byte array.
__concatenate	String	<p>The parameters can be literal strings or other values of various types. The result is a string that consists of all of these parameters concatenated together.</p> <pre>__concatenate("Active",protocol," Ports: ",portnum) __concatenate("CompanyName: [",CompanyName,"]") __concatenate("PF: ",PassOrBlock)</pre>
__concatenateDeleting	String	The last parameter is a literal string containing a set of characters to delete. The other parameters can be literal strings or other values of various types. The result is a string that consists of all of these parameters (except the last) concatenated together, with the specified characters deleted from the non-literal parameters. For example, if the parameters are "Literal", "Foobar", and "r" (where the first and third parameters are literal), then the result would be "LiteralFooba". Note that the "r" in "Foobar" was deleted but the "r" in "Literal" was not.
__contains	Boolean	<p>This operation searches for one string within another and returns true if it is found and false otherwise. For example, like</p> <pre>__contains(stringInWhichToSearch, stringToFind)</pre>

Operation	Return Type	Definition and Comments
__containsFromList	Boolean	This operation tries to match a string (the first operand, which is searched in) with a list of comma-separated strings and returns true when a string match is found. Otherwise returns false. For example, __containsFromList(stringInWhichToSearch , firstStringToFind, secondStringToFind)
__convertMSDNSURL	String	This operation converts a Microsoft DNS URL in the form: (n)nchars(m)mchars(0) To a normal URL: nchars.mchars
__createLocalTimeStampFromSeconds MicrosZone	TimeStamp	The parameters are 2 long integer numbers and a string. The first parameter is the number of seconds since January 1, 1970, while the second is the number of microseconds within the second. These are combined into a TimeStamp. If the third parameter is a valid time zone name, the number of seconds is interpreted relative to January 1, 1970 in that time zone. Otherwise GMT is used. Some of the precision of the microseconds is currently lost.
__createLocalTimeStampFromGMT SecondsMillis	TimeStamp	The 2 parameters are each long integer numbers. The first is the number of seconds since January 1, 1970 GMT, while the second is the number of milliseconds within the second. They are combined into a TimeStamp. __ createLocalTimeStampFromGMTSecondsMillis(tv_sec,tv_msec)
__createLocalTimeStampFromGMT Second Nanoseconds	TimeStamp	The 2 parameters are each long integer numbers. The first is the number of seconds since January 1, 1970 GMT, while the second is the number of nanoseconds within the second. They are combined into a TimeStamp. Some of the precision of the nanoseconds is currently lost.
__createLocalTimeStampFrom NanoSeconds	TimeStamp	The parameter is a long integer number. It is the number of nanoseconds since January 1, 1970 GMT. It is converted into a TimeStamp. Some of the precision of the nanoseconds is currently lost.
__createLocalTimeStampFromNTP	TimeStamp	The parameter is a string. It should contain the number of seconds since January 1, 1970 GMT before a decimal point, and the number of microseconds after the decimal point. They are combined into a TimeStamp.
__createLocalTimeStampFromSeconds SinceEpoch	TimeStamp	The parameter is a single long integer number, which is the number of seconds since January 1, 1970 GMT. It is converted into a TimeStamp, with the fractional seconds set to zero. __createLocalTimeStampFrom SecondsSinceEpoch (srcTimestamp)

Operation	Return Type	Definition and Comments
__createOptionalTimeStamp FromString	TimeStamp	The parameters are two strings. The first string is date and time specified by default in the yyyy-MM-dd HH:mm:ss format. The second, optional parameter specifies the format for the first string if it needs to be different from the default. If the value of the first string is null, nothing is mapped. Otherwise the value is mapped using the format specified for the second parameter, if present, or the default format.
__createRuleFiringInfo	String	This operation takes an arbitrary number of parameters. Each can be either a literal string or a value of some other type. The result is simply the parameters concatenated together as a long string, with commas between the parameters. The parameters which are not literal strings are converted to strings.
__createSafeLocalTimeStamp	TimeStamp	The first parameter is a string, which is the date/time to parse, while the second is a literal string, which is the format (same style as the format for the Date, Time, and TimeStamp tokens). The string is parsed and returned as a TimeStamp. Most errors result in the current time being returned.
__createTimeStamp	TimeStamp	The first parameter is a Date and the second parameter is a Time. They are combined into a single TimeStamp and returned. Everything is assumed to be in local time. __createTimeStamp(date,time)
__createTimeStampByHexEncodedTime	TimeStamp	The parameter is a single string of 12 hexadecimal digits, with 2 each for year (0 means 1970), month (0-11), day (1-31), hour (0-23), minute (0-59), and second (0-59). The milliseconds are implicitly set to zero, and the numbers are interpreted as local time. The resulting TimeStamp is returned.
__createTimeStampByStartTimeElapsed	TimeStamp	The parameters are 2 strings. The first is the starting time in ddMMMyyyy hh:mm:ss format, while the second is an elapsed time in hh:mm:ss format. The result is a TimeStamp for the ending time, assuming the starting time is a local time.
__createTimeStampForOpsecStartTime	TimeStamp	The parameter is a single string in ddMMMyyyy HH:mm:ss format. It is parsed and the resulting TimeStamp, interpreted as being local time, is returned.
__createTimeStampStringFrom SecondsMicros	String	The parameters are 2 long integer numbers. The first parameter is the number of seconds since January 1, 1970 GMT, while the second is the number of microseconds within the second. These are combined into a TimeStamp and then into a string. Some of the precision of the microseconds is currently lost.
__currentTimestampInSeconds	Long	Any parameters are ignored. The current time, expressed as the number of seconds since January 1, 1970 GMT, is returned as a long integer.

Operation	Return Type	Definition and Comments
__divide	Integer	The first parameter is the numerator and the second parameter is the denominator. The result is an integer with the value of the numerator divided by the denominator, rounded to the nearest integer.
__doubleToAddress	IPAddress	This is the same as the numberToAddress operation except that the parameter is a double-precision floating-point number. __doubleToAddress(DestIP)
__extractNTDomain	String	The only parameter is a string. If it contains a back slash, the part of the string up to but not including that backslash is returned. Otherwise the entire string is returned.
__extractNTUser	String	The only parameter is a string in the form 'domain\user', where domain is an NT domain and user is an NT user name. The user name is returned. If there is no backslash in the string, it is returned unchanged.
__extractProtocol	String	The only parameter is a string. If the string contains any of the defined protocol strings (TCP, ICMP, UDP, IGMP, or RTSP), just that string is returned (the search is case-insensitive, and the first protocol found is returned). If none of the protocol strings is found, the whole string is returned.
__firstOfPositiveInteger	Integer	This operation takes an arbitrary number of integer number parameters. The first one which is positive is returned. If no positive parameter is found, null is returned.
__foundScanHostName	String	The host name is returned in most cases. The exception is if the string is "[Unknown]", in that case null is returned.
__getCVEStringFor	String	The only parameter is a string, which should be a CVE identifier. What is returned is "CVEId" where id is the identifier. Note that the separator character is a vertical bar.
__getDeviceDirection	Enumeration (Integer)	The only parameter is a string. If it is one of the defined inbound strings (e.g., "in" or "incoming"), then the inbound constant (0) is returned. If it is one of the defined outbound strings (e.g., "outbound" or "=>"), then the outbound constant (1) is returned. Otherwise the unknown constant (Integer.MIN_VALUE, -2147483648) is returned.
__getIPv4AddressEmbeddedInIPv6Address	IPAddress	The operation extracts and returns an IPv4 address embedded in an IPv6 address. The return parameter is an IPv4 address. The input parameter is an IPv6 address in byte array format. To assign the IPv4 address to an IPv4 address event field in a non-IPv6-aware parser: __getIPv4AddressEmbeddedInIPv6Address (__stringToIPv6Address("::ffff:10.14.11.140"))

Operation	Return Type	Definition and Comments
__getIPv6AddressFromHighLow	String	This operator takes two string parameters consisting of decimal numbers and returns a string representation of an IPv6 address. The numbers are a decimal representation of the first four and last four segments of the IPv6 address.
__getLongMACAddressByHexString	MacAddress	The parameter is a 12-character hexadecimal string, which is converted to a MAC address.
__getLongMACAddressByString	MacAddress	The only parameter is a string. It is a MAC address, which is a 6-part hexadecimal address separated by colons or dashes. It is returned.
__getManhuntPriority	String	The two parameters are both long integers, with the first representing the severity and the second representing the reliability. The result is a string containing the product of the two values, divided by 256.
__getNormalizedOS	String	The only parameter is a string. This string is looked up in a map that comes from an AUP file. If found, the result is returned. Otherwise a string of the form "/Operating System/param" is returned, where param is the parameter string, with any slashes replaced by dashes. For example, "OS/2" would become "/Operating System/OS-2" (unless OS/2 appeared in the os.mappings.csv map, in which case that value would be returned).
__getNotZeroPort	Integer	The only parameter is a string. If it is null, not a valid integer, or zero, then null is returned. Otherwise (it is a valid non-zero integer), the numeric value is returned.
__getOriginator	Enumeration (Integer)	The only parameter is a string. If the string is "Source", the result is the source constant (0). If the string is "Destination", the result is the destination constant (1). Otherwise the unknown constant (Integer.MIN_VALUE, -2147483648) is returned.
__getOriginatorFromSourcePort	Enumeration (Integer)	The parameters are an Integer (the port number) and a literal integer. If neither is null and the port is less than the limit specified in the second (literal) parameter, then the destination constant (1) is returned. Otherwise the source constant (0) is returned.
__getProtocolName	String	The only parameter is an Integer, which is converted into a string for the matching protocol, as defined in RFC 1700. If the parameter is null, null is returned. And if the parameter is out of range, then the number itself is returned as a string.
__getProtocolNameFromString	String	This operation is like the getProtocolName operation, except that the parameter is a string instead of an integer. If the string does not contain a valid integer, then the string is returned unchanged.

Operation	Return Type	Definition and Comments
__getSymantecNSPriority	String	The two parameters are both long integers, with the first representing the severity and the second representing the reliability. The result is a string containing the product of the two values, divided by 10.
__getTimeZone	String	The only parameter is a string. If the string does not represent a valid timezone, it returns null. If the string is in the general timezone format, it returns the passed parameter. If the string is an offset in the RFC 822 format (such as "-08:00"), the return string is found by offset into the "timezones" list in agent.properties. Valid RFC 822 formats that are not found in agent.properties will return a reasonable default string.
__getTrendMicroHostName	String	The single parameter is a string. If it is null, null is returned. If it contains a backslash, then the part before the backslash is returned. If it contains an '@' or a '.', null is returned. Otherwise, the original string is returned.
__getTrendMicroUser	String	The first parameter is a string. If it contains a backslash that is not the final character of the string, then the part after the backslash is returned. If it contains an '@' or a '.', null is returned. Otherwise, the second parameter (which is a string if specified) is returned if specified. A null is returned if the second parameter is not specified.
__getTypeEnumeration	(Integer)	The only parameter is a literal string. If it is "correlation" or "correlated", then the correlation constant (2) is returned. If it is "aggregated," then the aggregated constant (1) is returned. Otherwise the base constant (0) is returned. The comparisons are made case-insensitively.
__getVendor	String	This is a synonym for the stringConstant operation.
__getVulnerabilityCategory	String	The only parameter is a literal integer, which should be in the range 0 to 4. The values returned are: <ul style="list-style-type: none"> • /scanner/device/vulnerability for 0 • /scanner/device/openport for 1 • /scanner/device/user for 2 • /scanner/device/banner for 3 • /scanner/device/uri for 4
__getXForceStringFor	String	If the one string parameter is not null, it is returned with 'X-Force!' prepended to it. If it is null, then null is returned.

Operation	Return Type	Definition and Comments
__hexStringToAddress	IPAddress	<p>This is similar to the noDotStringFormatToAddress operation, except that the parameter is in hexadecimal. In other words, it should be 8 hexadecimal digits, where each set of 2 digits is a part of the IP address, zero-filled and with no dots. For example, "COA80AOC" would become the IP address 192.168.10.12.</p> <p>Use this operation only with IPv6-aware parsers for both IPv4 and IPv6 addresses.</p>
__hexStringToLong	Long	<p>The one string parameter represents a hexadecimal value. If it starts with '0x' or '\$', those are removed before parsing the value. The result is returned as a long integer.</p>
__hexStringToIPv6Address	IPAddress	<p>For non-IPv6-aware parsers, this operator takes as input a 32-character string consisting of hexadecimal digits and converts it to an IPv6 address. If the length is 8 characters, as it would be for an IPv4 address, the return value is null. Any other input size results in an exception.</p> <p>For IPv6-aware parsers, this operation is obsolete and should not be used.</p>
__hexStringToString	String	<p>The parameter is a single string, which should consist of hexadecimal digits. It is converted to an array of bytes (two hexadecimal digits per byte), which is then converted to a string using UTF-8 encoding (RFC 3629). If the input is null, the result is also null.</p>
__hourMinuteSecondsToSeconds	Long	<p>The parameter is a single string, in HH:mm:ss format. The duration is converted to seconds and returned.</p>
__ifAorBThenElse	String	<p>There are five parameters. Each can be either a literal string or a regular string (although other types are converted to strings). If the first parameter is equal to the second or the first parameter is equal to the third parameter, then the fourth parameter is returned. Otherwise, the fifth parameter is returned.</p>
__ifGreaterOrEqual	String	<p>The four parameters are strings. If either of the first two parameters is null, null is returned and an error is logged. Otherwise, those two parameters are parsed as integers and compared. Any parsing errors treat the value as zero. If the first parameter is numerically larger than the second, then the third parameter is returned. Otherwise, the fourth parameter is returned.</p>
__ifPositive	String	<p>There are three parameters. If the first (integer) operand is positive, return the second (string) operand; otherwise, return the third (string) operand.</p>

Operation	Return Type	Definition and Comments
__ifThenElse	String	There are four parameters. Each can be either a literal string or a regular string (although other types are converted to strings). The first two parameters are compared, and if they are equal, then the third parameter is returned as the result. Otherwise (if the first two parameters differ), the fourth parameter is returned.
__ifThenElseAddress	IPAddress	There are four parameters. The first two parameters are string. The first two parameters are compared, and if they are equal, then the third parameter is returned as the result. Otherwise (if the first two parameters differ), the fourth parameter is returned.
__ifTrueThenElse	String	There are three parameters. The first is a Boolean value (true or false), and if it is true, then the second parameter is returned; if the Boolean value is false, then the third parameter is returned.
__integerConstant	Integer	The parameter is a single literal integer, which is returned. If a literal string which is not a valid integer is passed instead, then null is returned.
__integerToLong	Long	The parameter is a single integer number, which is converted to a long integer number and returned. If the parameter is null, the returned value is too.
__length	Integer	This operation retrieves the length of the operand string.
__longToDot4QuadAddress	String	The parameter is a single long integer number, which is converted to an IP address in the same manner as for the numberToAddress operation, but is then converted to a 4-part dotted string. For example, 16909060 would become the string "1.2.3.4".
__longToInteger	Integer	The parameter is a single long integer number, which is converted to an integer number (possibly truncating it) and returned. If the parameter is null, the returned value is too.
__longToString	String	This operation returns the string representation of a long object. The optional second operand is the radix (integer, minimum value is 2). The optional third operand is the minimum length (integer, minimum value is 0), and the result will be left-padded with zeroes, if needed to achieve that minimum length. This is useful in making numbers comparable as strings.
__longToTimeStamp	TimeStamp	The parameter is a single long integer number, which is the number of milliseconds since January 1, 1970 GMT. It is converted into a TimeStamp.

Operation	Return Type	Definition and Comments
__noDot4QuadStringsToAddress	IPAddress	<p>The parameters are 4 strings, each of which is a decimal number, and in the normal order for IP addresses. For example, the strings "192", "168", "10", "12" would become the IP address 192.168.10.12.</p> <p>__noDot4QuadStringsToAddress (src_ip1,src_ip2,src_ip3,src_ip4)</p>
__noDotStringFormatTo Address	IPAddress	<p>The parameter is a single string of 12 decimal digits, where each set of 3 digits is a part of the IP address, zero-filled and with no dots. For example, "192168010012" would become the IP address 192.168.10.12.</p>
__numberToAddress	IPAddress	<p>The parameter is a single long integer number, which is converted to an IP address with the least significant byte of the number corresponding to the rightmost part of the address. For example, 16909060 would become the IP address 1.2.3.4.</p> <p>__numberToAddress(IPAddress)</p>
__oneOf	String	<p>This operation takes an arbitrary number of parameters. Each can be either a literal string or a regular string. The first one that is not null and not zero-length is returned.</p>
__oneOfAddress	IPAddress	<p>For non-IPv6-aware parsers, this operation returns only the first non-null IPv4 address. For IPv6-aware parsers, this operation returns the first non-null IPv4 or IPv6 address.</p>
__oneOfDateTime	TimeStamp	<p>The parameters are any number of TimeStamp tokens. The first token, which is not null, is returned.</p>
__oneOfHostName	String	<p>For non IPv6-aware parsers, this operation works like the oneOf operation, but any parameter which looks like an IP address (4 decimal numbers separated by 3 periods) is skipped.</p> <p>For IPv6-aware parsers, this operation works like the oneOf operation, but any parameter which looks like an IPv4 or IPv6 address is skipped.</p>
__oneOfInteger	Integer	<p>This works like the oneOf operation, but the result is then parsed as an integer number and returned. If the value is not a valid number, null is returned.</p>
__oneOfLong	Long	<p>This works like the oneOf operation, but the result is then parsed as a long integer number and returned. If the value is not a valid number, null is returned.</p>
__oneOfMac	MacAddress	<p>This works like the oneOf operation, but the result is then parsed as a MAC address (a six octet hexadecimal representation, separated by colons) and returned. For example, 00:08:74:4C:7F:1D. If the value is not a valid MAC address, null is returned.</p>
__oneOfNetBIOSName	String	<p>This works like the oneOf operation, except for the removal of one or two leading backslashes, if present, before returning the result.</p>

Operation	Return Type	Definition and Comments
__parseMultipleTimeStamp	TimeStamp	The first parameter is a timestamp value, passed as a string. If it is null, null is returned. Otherwise, the second and any additional parameters are constant time stamp formats (as defined for Java's SimpleDateFormat class). They are used to attempt to parse the first parameter. The result of the first one that works, without throwing an exception, is returned as a TimeStamp. If none of the formats works, an exception is thrown.
__parseMutableTimeStamp	TimeStamp	<p>The parameter is a single string, which can be in one of these formats:</p> <ul style="list-style-type: none"> • MMM dd HH:mm:ss • MMM dd HH:mm:ss.SSS zzz • MMM dd HH:mm:ss.SSS • MMM dd HH:mm:ss zzz • MMM dd yyyy HH:mm:ss • MMM dd yyyy HH:mm:ss.SSS zzz • MMM dd yyyy HH:mm:ss.SSS • MMM dd yyyy HH:mm:ss zzz <p>If this operation has been called before successfully, the same format is tried first. If one of the first four formats (which do not include a year) is used, then the year is changed as described for the setYearToCurrentYear operation. If no format works, a fatal error is written to the log and null is returned.</p>
__parseMutableTimeStampSilently	TimeStamp	This is the same as the _parseMutableTimeStamp operation, except that when no format works, no fatal error is written to the log.
__parseSignedLong	Long	This is the same as the safeToLong operation, except that a leading "+" sign is also allowed.
__product	Integer	Each parameter is either an integer variable or a string constant that can be a floating-point value. The result is an integer with the value of the product of the parameters multiplied together and rounded to the nearest integer.
__regexToken	String	<p>This operation takes two strings as parameters. The first is the string to parse. The second is the regular expression (a literal string). If the regular expression is blank or null then the result is the same as the first argument. Otherwise the string to parse is parsed using the regular expression, and the first matching group (expression inside parentheses) is returned as a string. For example, if the parameters are "foobar" and "fo+(o.*)X(r)", the result will be "oba".</p> <p>__regexToken(proto, ".*/(.*)")</p>

Operation	Return Type	Definition and Comments
__regexTokenAsAddress	IPAddress	<p>For non-IPv6-aware parsers, this operation is similar to the regexToken operation: it takes two string parameters, and the result (expected to be in four-part dotted decimal format) is then converted from a string to an IP address. That is, if the parameters are "foo/192.168.10.12/bar" and "[a-z]+\V([0-9\.]++)\V/bar", the result will be the IP address 192.168.10.12.</p> <p>__regexTokenAsAddress (dst,"(.*)[:].*")</p> <p>For IPv6-aware parsers, this operation can return both IPv4 and IPv6 addresses.</p>
__regexTokenAsInteger	Integer	<p>This is like the regexToken operation, also taking 2 string parameters, except that the result is then converted from a string to an integer (or null if it is not a valid number).</p> <p>__regexTokenAsInteger (port,".*?:((\d+))")</p> <p>__regexTokenAsInteger (dst,".*?:((\d+))[:].*")</p>
__regexTokenAsLong	Long	<p>This is like the regexToken operation, also taking 2 string parameters, except that the result is then converted from a string to a long integer (or null if it is not a valid number).</p>
__regexTokenFindAndJoin	String	<p>There are five string parameters. The first parameter is the string to be processed. The second is a regular expression with at least one capturing group. The third is an optional join delimiter. The fourth and fifth are optional strings to prepend and append to the final result, respectively. The operation repeatedly attempts to find the regular expression in the string to be processed, starting each time at the end of where the regular expression was last found. Each time it is found, the capturing groups from the regular expression are added to the result, with the join delimiter between them. Finally, the prepend and append strings are added, if they are not null.</p>
__regexTokenNoWarning	String	<p>This operation works similarly to the regexToken operation. The primary differences are that 1) the regular expression has to match the entire string, not just be found in it, and 2) if the regular expression does not match, there is no warning logged.</p>
__replaceAll	String	<p>The three parameters are all strings. The first is the starting string, the second is the regular expression, and the third is the replacement string. Each place the regular expression is found in the starting string is replaced by the replacement string, and the result is returned. Note that the replacement string can contain references to capturing groups in the regular expression, in the form '\$n', where n is 0 to 9.</p>

Operation	Return Type	Definition and Comments
__replaceFirst	String	The three parameters are all strings. The first is the starting string, the second is the regular expression, and the third is the replacement string. The first place the regular expression is found in the starting string it is replaced by the replacement string, and the result is returned. Note that the replacement string can contain references to capturing groups in the regular expression, in the form '\$n', where n is 0 to 9.
__reverseDottedDecimalAddress ByteOrder	String	The parameter is an IP address passed as a string, which must have exactly 3 dot characters. The result is an IP address returned as a string, but with the 4 parts reversed in order. For example, passing '2.1.168.192' will result in '192.168.1.2' being returned.
__safeToDate	TimeStamp	This operation works like the createOptionalTimeStampFromString operation, except that if errors occur, null is returned.
__safeToInteger	Integer	The parameter is a single string, which is converted to an integer, or null if the string is not a valid number. Useful for log formats that use "-" to specify null values on integer fields, such as Microsoft Windows XP SP2 Personal Firewall. __safeToInteger(bytes) __safeToInteger(srcPort)
__safeToLong	Long	The parameter is a single string, which is converted to a long integer, or null if the string is not a valid number. __safeToLong(time_taken)
__safeToRoundedLong	Long	The parameter is a string that is parsed as a number (which can have a fractional part) and then rounded to the nearest long integer and returned. If the string is not a valid number, null is returned.
__setYearToCurrentYear	TimeStamp	The parameter is a single TimeStamp, for which the year is forcibly set to the current year, plus or minus one (depending in part on the syslog.future.limit property). This is used for TimeStamps that do not have a defined year.
__signedNumberToAddress	IPAddress	The parameter is a long integer that is returned as an IP address, but with the byte-order reversed.

Operation	Return Type	Definition and Comments
__simpleMap	String	<p>There are n+1 or n+2 parameters. The first parameter is a string which is to be looked up in the map. The next n parameters are the map, in the form of string literals each of which has a key, an equals sign, and a value. If the key matches the first parameter, then the value for that key is returned. If the final parameter is a single character, it is used as the delimiter instead of the equals sign. For example, if the parameters are (all literal except the first): "Foo", "Bar=17", "Foo=34", then the returned value will be "34". If no key matches, null is returned.</p> <p>__simpleMap(FileInfected,"0=No", "1=Yes", "=") __simpleMap(Type,"8=Success", "16=Failure")</p>
__split	String	<p>This operation takes three parameters. The first is the string to split (a string). The second is the delimiter (a literal string). The third is the index (a literal integer). If the delimiter or the index is blank or null, then the result is the same as the first argument. Otherwise the string to split is split around occurrences of the delimiter, with the index'th string returned. For example, if the parameters are "The string to split," " " (space), and "2", the result will be "string".</p>
__splitAsAddress	IPAddress	<p>For non-IPv6-aware parsers, this operation is like the split operation: it takes three string parameters, and the result (expected to be in four-part dotted decimal format) is then converted from a string to an IP address. That is, if the parameters are "foo/192.168.10.12/bar", "/", and 2, the result will be the IP address 192.168.10.12.</p> <p>For IPv6-aware parsers, this operation converts the result to an IPv4 or IPv6 address.</p>
__splitAsInteger	Integer	<p>This is like the split operation, also taking 3 string parameters, except that the result is then converted from a string to an integer (or null if it is not a valid number).</p>
__splitAsLong	Long	<p>This is like the split operation, also taking 3 string parameters, except that the result is then converted from a string to a long integer (or null if it is not a valid number).</p>
__stringConstant	String	<p>This takes a single string literal parameter, and returns it.</p> <p>__stringConstant("Example")</p>
__stringToIPv6Address	IPAddress	<p>In a non-IPv6-aware parser, this operation takes a string representation of an IPv6 address as input and returns a value of type IPv6 address.</p> <p>This operation should not be used in a IPv6-aware parser. Instead, use the IP Address token parser or directly map the IPv6 address string to event fields.</p>
__stringTrim	String	<p>The parameter is a string, that is returned with any leading or trailing whitespace characters removed.</p>

Operation	Return Type	Definition and Comments
__subtract	Integer	The two parameters must be integer variables, or can be string constants that are floating-point values. The result is an integer with the value of the first parameter minus the second and rounded to the nearest integer.
__sum	Integer	Each parameter must be an integer variable, or can be a string constants that are floating-point values. The result is an integer with the value of the sum of the parameters added together and rounded to the nearest integer.
__toHex	String	The parameters are a long integer number and a literal integer. The value of the first parameter is converted to hexadecimal and returned, padded to the number of digits specified by the second parameter, and preceded by "0x". Note that odd lengths are rounded down, and if the specified length is insufficient some of the bits of the first parameter are simply lost. For example, with parameters of 65535 and 8, the result is "0x0000FFFF". With parameters of 65535 and 3, the result is "0xFF" (the 3 is rounded down to 2, and the high-order bits of 65535 are lost).
__toLongTimeStamp	Long	The parameter is a single string, which is a date and time in yyyy-MM-dd HH:mm:ss format. The string is parsed, interpreting it as local time, and the resulting date is returned as the long integer number of milliseconds since January 1, 1970 GMT.
__toLowerCase	String	The parameter is a single string, which is converted to lowercase and returned. __toLowerCase(protocol)
__toUpperCase	String	The parameter is a single string, which is converted to uppercase and returned. __toUpperCase(protocol)
__useCurrentYear	TimeStamp	The parameter is a single TimeStamp, which is returned with its year changed to the current year. The calculation is done in the local timezone, which will affect the result near either end of the year. __useCurrentYear(date)

Appendix B: ArcSight Built-in Tokens

This table lists ArcSight built-in tokens

Token String	Description
Tokens Available for Database Parsers Only	
_DB_DRIVER	JDBC Driver Name.
_DB_URL	Database URL.
_DB_HOST	Host name or IP Address of the machine hosting the database.
_DB_PORT	Port where the database is listening for SQL queries.
_DB_NAME	Database name.
Tokens Available for Syslog Parsers Only	
_SYSLOG_TIMESTAMP	Time stamp received in the header of the syslog message.
_SYSLOG_SENDER	Host name or IP address of the sender received in the header of the syslog message. In the unusual case if the header did not contain a host name or IP address, this will be the address that the connector received the packet from.
_SYSLOG_SOURCE_ADDR	The actual IP address that the connector received the syslog message from. The token value can be assigned to the event field of your choice. (For example, event.deviceCustomString6=_SYSLOG_SOURCE_ADDR). The value of this token can be an IPv4 or an IPv6 address.
_SYSLOG_FACILITY	Facility received in the header of the syslog message (applies only to Syslog Daemon connector).
_SYSLOG_PRIORITY	Priority received in the header of the syslog message (applies only to Syslog Daemon connector).
Tokens Available for SNMP Parsers Only - the following token strings are for use only in the properties file type <code>sdksnmp.#.sdksnmptrap.properties</code>	
_SNMP_TRAP_TYPE	Trap type received in the SNMP trap header.
_SNMP_TIMESTAMP	Time stamp received in the SNMP trap header.
_SNMP_ENTERPRISE_OID	Enterprise OID received in the SNMP trap header.
_SNMP_SENDER	SNMP agent address that sends the SNMP trap.

Appendix C: ArcSight Built-in Token Types

Token types are important because tokens can only be mapped to ArcSight event fields with matching types. Event fields and their types are listed in the *ArcSight Console User's Guide*, in the "Reference Guide", under "Data Fields".

Type	Meaning	Format
Date	A value evaluating to a particular day.	MM/dd/yyyy
Integer	A number from -2147483648 to 2147483647.	n/a
IPAddress	For non-IPv6-aware parsers, this is an IPv4 address (for example: 1.1.1.1). This type cannot be used for IPv6 addresses. If it is, then null will be returned. For IPv6-aware parsers, this can be an IPv4 or an IPv6 address (for example: fdbf59b2e1356c9:xxxx:xxxx:xxxx:xxxx).	n/a
IPv6Address	An IPv6 address - 16 bytes specified as 32 hexadecimal characters where each byte consists of two hexadecimal characters.	n/a
Long	A number from -9223372036854775808 to 9223372036854775807.	n/a
MacAddress	An Ethernet MAC address of the form: 00-06-3E-22-51-B9 or 00:06:3E:22:51:B9.	n/a
RegexToken	This token type is useful when a simple regular expression needs to be used to extract further information from a token. For example: Assume that the token contained the string 'From: rajiv' and the only needed part is 'rajiv' then the following expression could be used: s/From: (.*)/\$1/	Substitution regular expression of the form: s/{exp}/{subst}/
String	Any free form sequence of characters.	n/a
Time	A value evaluating to a particular time of day.	HH:mm:ss
TimeStamp	A date, a time or a date and a time.	Date/time format (see Date and Time Format Symbols)

Appendix D: Date and Time Format Symbols

This table contains date and time format symbols:

Symbol	Meaning	Presentation	Examples
G	Era designator	(Text)	AD
y	Year	(Number)	2016 or 16
M	Month in year	(Text & Number)	July or Jul or 07
w	Week in year	(Number)	27
W	Week in month	(Number)	2
D	Day in year	(Number)	129
d	Day in month	(Number)	10
F	Day of week in month	(Number)	2 (indicating 2nd Wed. in July)
E	Day in week	(Text)	Tuesday or Tue
a	Am/pm marker	(Text)	AM or PM
H	Hour in day (0~23)	(Number)	0
k	Hour in day (1~24)	(Number)	24
K	Hour in am/pm (0~11)	(Number)	0
h	Hour in am/pm (1~12)	(Number)	12
m	Minute in hour	(Number)	30
s	Second in minute	(Number)	55
S	Millisecond	(Number)	978
z	Time zone	(Text)	Pacific Standard Time or PST or GMT-08:00
Z	Time zone	RFC 822	-0800 (indicating PST)

For example, one date format might be:

yyyy-MM-dd HH:mm:ss

Use single quotes around text that is not meant to be interpreted as date format characters. Use this example for a date like: 2016.07.04 AD at 12:08:56 PDT.

yyyy.MM.dd G 'at' HH:mm:ss z

Use two single quotes to insert a single quote. Use this example for a date like: Wed, Jul 4, '16.

EEE, MMM d, ''yy

Appendix E: ArcSight Built-in Event Field Mappings

The following table lists ArcSight event fields. See the numbered Range Notes (n) following this table for further explanations of certain field ranges.

ArcSight Mapping	Type	Length	Range
applicationProtocol	String	31	n/a
baseEventCount	Integer	n/a	0 -> 2 ³¹ -1
bytesIn	Long	n/a	0 -> 2 ⁶³ -1
bytesOut	Long	n/a	0 -> 2 ⁶³ -1 -1
categoryBehavior	String	1023	n/a (1)
categoryDeviceGroup	String	1023	n/a (1)
categoryObject	String	1023	n/a (1)
categoryOutcome	String	1023	n/a (1)
categorySignificance	String	1023	n/a (1)
categoryTechnique	String	1023	n/a (1)
cryptoSignature	String	512	n/a
customerURI	String	-	n/a (2)
destinationAddress	IPAddress	n/a	IPv4 or IPv6 (3)
destinationDnsDomain	String	255	n/a
destinationHostName	String	1023	n/a
destinationMacAddress	MacAddress	n/a	MAC (4)
destinationNtDomain	String	255	n/a
destinationPort	Integer	n/a	0 -> 65535
destinationProcessName	String	1023	n/a
destinationServiceName	String	1023	n/a
destinationTranslatedAddress	IPAddress	n/a	IPv4 or IPv6 (3)
destinationTranslatedPort	Integer	n/a	0 -> 65535
destinationTranslatedZoneURI	String	-	n/a (2)

ArcSight Mapping	Type	Length	Range
destinationUserId	String	1023	n/a
destinationUserName	String	1023	n/a
destinationUserPrivileges	String	1023	n/a
destinationZoneURI	String	-	n/a (2)
deviceAction	String	63	n/a
deviceAddress	IPAddress	n/a	IPv4 or IPv6 (3)
deviceCustomDate1	TimeStamp	n/a	n/a (5)
deviceCustomDate1Label	String	1023	n/a
deviceCustomDate2	TimeStamp	n/a	n/a (5)
deviceCustomDate2Label	String	1023	n/a
deviceCustomIPv6Address1	IPAddress	n/a	IPv6 (8)
deviceCustomIPv6Address1Label	String	1023	Should be "Device IPv6 Address"
deviceCustomIPv6Address2	IPAddress	n/a	IPv6 (8)
deviceCustomIPv6Address2Label	String	1023	Should be "Source IPv6 Address"
deviceCustomIPv6Address3	IPAddress	n/a	IPv6 (8)
deviceCustomIPv6Address3Label	String	1023	Should be "Destination IPv6 Address"
deviceCustomNumber1	Long	n/a	- 2 ⁶³ -> 2 ⁶³ -1
deviceCustomNumber1Label	String	1023	n/a
deviceCustomNumber2	Long	n/a	- 2 ⁶³ -> 2 ⁶³ -1
deviceCustomNumber2Label	String	1023	n/a
deviceCustomNumber3	Long	n/a	- 2 ⁶³ -> 2 ⁶³ -1
deviceCustomNumber3Label	String	1023	n/a
deviceCustomString1	String	1023 (4.x) 4000 (5.x)	n/a
deviceCustomString1Label	String	1023	n/a
deviceCustomString2	String	1023 (4.x) 4000 (5.x)	n/a
deviceCustomString2Label	String	1023	n/a
deviceCustomString3	String	1023 (4.x) 4000 (5.x)	n/a

ArcSight Mapping	Type	Length	Range
deviceCustomString3Label	String	1023	n/a
deviceCustomString4	String	1023 (4.x) 4000 (5.x)	n/a
deviceCustomString4Label	String	1023	n/a
deviceCustomString5	String	1023 (4.x) 4000 (5.x)	n/a
deviceCustomString5Label	String	1023	n/a
deviceCustomString6	String	1023 (4.x) 4000 (5.x)	n/a
deviceCustomString6Label	String	1023	n/a
deviceDnsDomain	String	255	n/a
deviceDomain	String	1023	n/a
deviceEventCategory	String	1023	n/a
deviceEventClassId	String	1023	n/a
deviceExternalId	String	255	n/a
deviceFacility	String	1023	n/a
deviceHostName	String	63	n/a
deviceInboundInterface	String	15	n/a
deviceMacAddress	MacAddress	n/a	MAC(4)
deviceNtDomain	String	255	n/a
deviceOutboundInterface	String	15	n/a
devicePayloadId	String	128	n/a
deviceProcessName	String	1023	n/a
deviceProduct	String	63	n/a
deviceReceiptTime	TimeStamp	n/a	n/a (5)
deviceSeverity	String	63	n/a
deviceTimeZone	String	255	n/a
deviceTranslatedAddress	IPAddress	n/a	IPv4 or IPv6 (3)
deviceTranslatedZoneURI	String	-	n/a (2)
deviceVendor	String	63	n/a
deviceVersion	String	31	n/a

ArcSight Mapping	Type	Length	Range
deviceZoneURI	String	-	n/a (2)
endTime	TimeStamp	n/a	n/a (5)
externalId	String	40	n/a
fileCreateTime	TimeStamp	n/a	n/a (5)
fileHash	String	255	n/a
fileId	String	1023	n/a
fileModificationTime	TimeStamp	n/a	n/a (5)
fileName	String	1023	n/a
filePath	String	1023	n/a
filePermission	String	1023	n/a
fileSize	Long	n/a	0 -> 2 ⁶³ -1
fileType	String	1023	n/a
flexDate1	TimeStamp	n/a	n/a (5)
flexDate1Label	String	128	n/a
flexNumber1	Long	n/a	-2 ⁶³ -> 2 ⁶³ -1
flexNumber1Label	String	128	n/a
flexNumber2	Long	n/a	-2 ⁶³ -> 2 ⁶³ -1
flexNumber2Label	String	128	n/a
flexString1	String	1023	n/a
flexString1Label	String	128	n/a
flexString2	String	1023	n/a
flexString2Label	String	128	n/a
message	String	1023	n/a
name	String	512	n/a (9)
oldFileCreateTime	TimeStamp	n/a	n/a (5)
oldFileHash	String	255	n/a
oldFileId	String	1023	n/a
oldFileModificationTime	TimeStamp	n/a	n/a (5)
oldFileName	String	1023	n/a
oldFilePath	String	1023	n/a

ArcSight Mapping	Type	Length	Range
oldFilePermission	String	1023	n/a
idFileSize	Long	n/a	0 -> 2 ⁶³ -1
idFileType	String	1023	n/a
rawEvent	String	4000	n/a (7)
requestClientApplication	String	1023	n/a
requestContext	String	2048	n/a
requestCookies	String	1023	n/a
requestMethod	String	1023	n/a
requestUrl	String	1023	n/a
sourceAddress	IPAddress	n/a	IPv4 or IPv6 (3)
sourceDnsDomain	String	255	n/a
sourceHostName	String	1023	n/a
sourceMacAddress	MacAddress	n/a	MAC (4)
sourceNtDomain	String	255	n/a
sourcePort	Integer	n/a	0 -> 65535
sourceProcessName	String	1023	n/a
sourceServiceName	String	1023	n/a
sourceTranslatedAddress	IPAddress	n/a	IPv4 or IPv6 (3)
sourceTranslatedPort	Integer	n/a	0 -> 65535
sourceTranslatedZoneURI	String	-	n/a (2)
sourceUserId	String	1023	n/a
sourceUserName	String	1023	n/a
sourceUserPrivileges	String	1023	n/a
sourceZoneURI	String	-	n/a (2)
startTime	TimeStamp	n/a	n/a (5)
transportProtocol	String	31	n/a (6)

Range Notes

- Although these fields can be set using the FlexConnector properties file, the recommended way is to create a categorization file. For more about the possible values, see the "Categories" topic in the Console Help or the *ArcSight Console User's Guide*. Also, see ["FlexConnectors and Categorization"](#).
- Although URI fields can be set using the FlexConnector properties file, these are really links to

resources in the database. Therefore, it is recommended that those fields be set using the network-model and customer-setting features.

3. This can be an IPv4 address (from **0.0.0.0** to **255.255.255.255**) or an IPv6 address (**xxxx:xxxx:xxxx:xxxx:xxxx:xxxx**).
4. This is a MAC address: **XX:XX:XX:XX:XX:XX** or **XX-XX-XX-XX-XX-XX**.
5. This is a timestamp stored as milliseconds since January 1, 1970.
6. The options are: TCP, UDP, ICMP, IGMP, ARP.
7. Set **PreserveRawEvent** to **Yes** to have the connector automatically preserve the original event log received from the device. With the default **No**, you can configure this field. To find the **PreserveRawEvent** field in the ArcSight Console interface, go to the **Connectors resource tree > Configure > Default tab > Content > Processing section > PreserveRawEvent**.
8. For a non-IPv6-aware parser, the IPv6 fields (**deviceCustomIPv6Address1**, **2**, and **3**) should consistently use 1 for device, 2 for source, and 3 for destination. The labels for them will automatically be set if the IPv6 address field is set, but if your ArcSight Console parser sets them explicitly, it should use the exact strings shown above.

For an IPv6-aware parser, the IPv6 fields (**deviceCustomIPv6Address1**, **2**, and **3**) can contain either IPv4 or IPv6 addresses. In practice, these fields should rarely be used. If they are, the labels should be set to an appropriate value.

9. The **name** field is mandatory.

See "[ArcSight Built-in Tokens](#)" for a list of ArcSight built-in tokens.

Appendix F: Configuring a Connector for ArcSight ESM Domain Field Sets

This appendix applies to Oracle-based ESM and provides information on configuring a FlexConnector for ESM domain field sets, which allow you to map additional data.

ArcSight ESM offers a series of special user-configurable fields called domain fields that you can use to leverage additional data available in an event, and that identifies a business-related attribute. When events come in to the ArcSight Manager, they are evaluated against the available domain field sets. If the event matches the fields in a domain field set, the event is tagged as relevant to that domain. These fields then are displayed in the Event Inspector and anywhere that domain field set is referenced.

Before creating a FlexConnector or modifying an existing connector to send additional data to support domain field sets, create the domain fields and domain field sets from the ArcSight Console as described in **Domain Field Sets** in the *ArcSight ESM User's Guide*.

Supported data types include:

FlexConnector Data Type	ESM Data Type
String	String
Long	Long
TimeStamp	Date
IPAddress (IPv4)	IPv4Address
Integer	Number
IPv6Address	IPv6Address
Double (Floating Point)	Floating Point

You can modify an existing FlexConnector or create a new FlexConnector to take advantage of the fields you have defined as part of the domain field set. For example, the following is a domain field set for credit card transactions:

Field	Type
Credit Card Number	Integer (Number)
Transaction Amount	Double (Floating Point)
Currency	String
Transaction Host IP	IPv6Address
Transaction Time	TimeStamp (Date)

Assuming these fields are not defined in a current parser, you will need to add mappings for these fields in the FlexConnector parser as additional data fields.

For example, for this sample domain field set, you can add the following entries to the FlexConnector parser you are developing:

```
token[0].name=Credit Card Number
token[0].type=Integer
```

```
token[1].name=Transaction Amount
token[1].type=Double
```

```
token[2].name=Currency
token[2].type=String
```

```
token[3].name=Transaction Host IP
token[3].type=IPAddress
```

```
token[4].name=Transaction Time
token[4].type=TimeStamp
```

```
additionaldata.Credit Card Number=Credit Card Number
additionaldata.Transaction Amount=Transaction Amount
additionaldata.Currency=Currency
additionaldata.Transaction Host IP=Transaction Host IP
additionaldata.Transaction Time=Transaction Time
```

The connector processes the additional data fields with the data type you assigned along with the token names.

If you have an existing FlexConnector, you can modify your parser to include the new fields for a domain field set as shown in the following example. To modify the parser of an existing SmartConnector that you have installed, contact Professional Services or your ArcSight representative for assistance.

The domain field set for this example includes the following fields:

Field	Type
Credit Card Number	Integer (Number)
Credit Card Holder	String
Transaction Host IP	IPAddress (for IPv6-aware parsers, this can be an IPv4 or IPv6 address)
Transaction Time	TimeStamp (Date)

In this example, your existing parser contains entries such as the following:

```
token[9].name=abc  
token[9].type=TimeStamp
```

```
token[10].name=def  
token[10].type=Integer
```

```
token[11].name=ghi  
token[11].type=IPAddress
```

```
token[12].name=jkl  
token[12].type=String
```

```
token[13].name=mno  
token[13].type=Long
```

You can use currently defined tokens to assign data types to your new domain feature set fields by adding these additional data fields. Transaction Time will assume the data type of the **jkl** field (**TimeStamp**).

When you add this...	Then...
additionaldata. Credit Card Number =abc	Credit Card Number assumes the data type of the abc field (Integer)
additionaldata. Credit Card Holder =def	Credit Card Holder assumes the data type of the def field (String)
additionaldata. Transaction Host IP =ghi	Transaction Host IP assumes the data type of the ghi field (IPAddress)
additionaldata. Transaction Time =jkl	Transaction Time assumes the data type of the jkl field (TimeStamp)

These additional data fields associate your newly created fields with the data types of fields already defined in the parser.

After modifying the parser, restart the connector. When the connector comes back online, it sends the added fields to the ArcSight Manager.

Appendix G: Advanced Parameters

The following topics are covered in this appendix:

- [Parameters Common to all SmartConnectors](#)
- [CEF Syslog Parameters](#)
- [File Connector Parameters](#)
- [File Folder Follower Parameters](#)
- [Syslog Parameters](#)

Note:

- The advanced parameters have been designed to assist developers in creating new FlexConnectors. The advanced parameters might not be applicable to all connectors even if they are present in the **agent.properties** file. If they are applicable to a connector, they will work as described.
- Do not change any parameter value in the **agent.properties** file unless the parameter is described in your connector's guide. This appendix is meant for developing new FlexConnectors, and not for changing parameters in the implemented connectors. Changing the parameters from their default values can prevent the connectors from working.

You can customize connector behavior by using the advanced parameters described in this appendix. These parameters can be added to or updated in the **agent.properties** file located in the **\$ARCSIGHT_HOME/current/user/agent** directory after connector installation.

Note: The folder path examples in this chapter refer to the Linux form where the path starts with **\$ARCSIGHT_HOME** and uses slashes. For Windows, the path starts with **%ARCSIGHT_HOME%** and uses back slashes. For example:

- Linux: **\$ARCSIGHT_HOME/current/user/agent/agent.properties**
- Windows: **%ARCSIGHT_HOME%\current\user\agent\agent.properties**

The **agent.properties** file is a plain text file. Use the appropriate editor for your operating system to edit the content. For example, use Notepad for Windows and vi for Linux. Any modifications to the **agent.properties** file should be performed very carefully with knowledge of how parameters operate. This way, you would avoid inadvertently altering the behavior of the SmartConnector.

Parameters Common to all SmartConnectors

The following table describes the parameters that can be used with all ArcSight SmartConnectors.

Parameter	Default	Description
agents[x].deviceconnection alertinterval	60000	Connectors update internal device connectivity state based on this interval (milliseconds).
agents[x].extractfieldnames	[blank]	<p>List of event fields separated by comma; for example: fileName, sourcePort.</p> <p>This parameter is related to fieldextractor feature that allows to populate the event field based on the file name the connector is reading. It can be used in all connectors that have files as input (for example any file connector, file folder connector and a few other connectors like DB Audit processing, Juniper Steel-Belted Radius).</p> <p>extractfieldnames, extractregex and extractsource are used together. Only when usefieldextractor is true, extractfieldnames, extractregex and extractsource can be used.</p>
agents[x].extractregex	[blank]	<p>The regular expression that will extract as many tokens (filename, sourcePort, and so on) as the number of fieldnames from the name of the log file.</p> <p>This parameter is related to fieldextractor feature that allows to populate the event field based on the file name the connector is reading. It can be used in all connectors that have files as input (for example any file connector, file folder connector and a few other connectors like DB Audit processing, Juniper Steel-Belted Radius).</p> <p>extractfieldnames, extractregex and extractsource are used together. Only when usefieldextractor is true, extractfieldnames, extractregex and extractsource can be used.</p>

Parameter	Default	Description
agents[x].extractsource	File Name	<p>Source from which to extract the fields.</p> <p>Possible Values: A constant—"File Name" or "File Path".</p> <p>This parameter is related to fieldextractor feature that allows to populate the event field based on the file name the connector is reading. It can be used in all connectors that have files as input (for example any file connector, file folder connector and a few other connectors like DB Audit processing, Juniper Steel-Belted Radius).</p> <p>extractfieldnames, extractregex and extractsource are used together. Only when usefieldextractor is true, extractfieldnames, extractregex and extractsource can be used.</p>
agents[x].persistenceinterval	0	<p>Interval in milliseconds when persisting properties in some connectors.</p> <ul style="list-style-type: none"> The persisted file is located at <code>\$ARCSIGHT_HOME/current/user/agent/persisted.properties</code>. If 0, then every change in the property value will persist in the file. This could impact the performance. If < 0, then the properties file is not persisted. If > 0, then wait for the specified interval and then persist properties file. The property file will have the properties specified by the connector. For example, for legacy connectors it could contain the file name as a key, and "true" as value if file was processed.
agents[x].unparsedevents.log.enabled	false	<p>The default value is false. Specify true for the connector to detect and log unparsed events to <code>\$ARCSIGHT_HOME/current/logs/events.log</code>. See also "Unparsed Events Detection".</p>
agents[x].usefieldextractor	false	<p>Indicates whether the event fields should be extracted from the log file name.</p> <p>Possible Values: true/false</p> <p>This parameter is related to fieldextractor feature that allows to populate the event field based on the file name the connector is reading. It can be used in all connectors that have files as input (for example any file connector, file folder connector and a few other connectors like DB Audit processing, Juniper Steel-Belted Radius).</p>

Parameter	Default	Description
deviceeventcounter.maxdevicestoevent	1000	This property pertains to the DeviceEventCounter module. It specifies the default maximum number of devices for which the DeviceEventCounter module will send agent:043 internal events. If an agent properties file specifies agent.component[x].maxdevstoevent as a legacy parameter, then it will be used instead of the deviceeventcounter.maxdevicestoevent value.
deviceeventcounter.maxdevicestolog	1000	This property pertains to the DeviceEventCounter module. It specifies the default maximum number of devices for which the DeviceEventCounter module will log the EPS status. If an agent properties file specifies agent.component[x].maxdevicestolog as legacy parameter, then it will be used instead of the deviceeventcounter.maxdevicestolog value.
name.resolve.use.getallbyname	true	Flag to control if java.net.getAllByName is used (if false then getByName is used, which may avoid IPv6 lookups).

CEF Syslog Parameters

The following table describes the CEF syslog parameters.

Parameter	Default	Description
transport.cefsyslog.header	false	Change to true to enable RFC 3164 headers for the CEF Syslog destination type.
transport.cefsyslog.header.facility	4	This parameter, which is ignored unless transport.cefsyslog.header is true, changes the facility value used to calculate the <PRI> value in the generated header. The range of valid values is 0 to 23. The default value of 4 means "security/authorization messages".

Parameter	Default	Description
transport.cefsyslog.header.keepdomain	false	This parameter, which is ignored unless transport.cefsyslog.header is true, controls whether the value in deviceHostname is used as is in the header (if the property changed to true), or if the domain is first removed. For example, if the deviceHostName field of an event is server.foo.com, only "server" would normally be used in the header. But if this property is changed to true, then "server.foo.com" would be used. Note that for any events that do not have the deviceHostName field set, this property does not matter (the deviceAddress will be used instead).
transport.cefsyslog.header.severitymap	7,6,5,3,2	This parameter, which is ignored unless transport.cefsyslog.header is true, controls how the event's agentSeverity field is converted into an RFC 3164 severity value, which in turn is combined with the facility value to create the <PRI> value in the generated header. If this property is changed, there must be 5 values, representing agentSeverity values unknown, low, medium, high, and very-high, respectively. And each value must be in range of 0 (emergency) to 7 (debug). The default mapping is unknown=> debug, low=> informational, medium=> notice, high=> error, and very-high=> critical.
transport.cefsyslog.header.useconadrashost	true	This parameter, which is ignored unless transport.cefsyslog.header is true, controls what to do if neither the deviceHostname nor the deviceAddress field is set in an event. By default the connector's own IP address is used, but that can be disabled (leaving that part of the header empty) by changing this property to false.

File Connector Parameters

The following table describes the file connector parameters.

Parameter	Default	Description
agents[x].configfile	Agent_ type/Agent_ type	Path of the config file. This is the directory where the connector gets a parser for log(s).
agents[x].followexternal rotation	false	<p>If property is set to false, the rotation "in place" is not followed, the file read once, the drop in size may not be monitored. If set to true, it will be monitored. There are different ways files are rotated:</p> <ul style="list-style-type: none">• If the new file has "index" suffix. For example, log1.txt,log2.txt• If the new file has a new date stamp added. For example, log2013-07-31.txt for daily rotation. <p>So, these types of rotations are captured with "rotationscheme" and related parameters, or some rotation specified directly in file name regex, like those based on multifolderfollower (Apache Tomcat is an example).</p> <p>If this property is set to true, the Connector will monitor the size of the file (using the file's name, not its inode). If the file size has decreased, the connector will assume that the file has been rotated.</p>
agents[x].internalevent. filecount.enable	false	<p>Enable/disable internal events when the number of files processed does not meet the user defined limits.</p> <ul style="list-style-type: none">• agents[x].internalevent.filecount.duration=n Specifies the number of seconds.• agents[x].internalevent.filecount.minfilecount=n Specifies the minimum number of files that the connector should process in the duration specified.• agents[x].internalevent.filecount.timer.delay=n Specifies, in seconds, the time the SmartConnector waits after it starts monitoring and sending internal events when needed.
agents[x].internalevent. fileend.enable	true	Sends an internal event when file has completed processing.
agents[x].internalevent. filestart.enable	true	Sends an internal event when file has started to process.
agents[x].logfilename	[blank]	<p>This property will be interpreted as a directory/folder. For example:</p> <p>logfilename=/home/logfiles/</p>

Parameter	Default	Description
agents[x].onrotation	None	<p>Possible Values:</p> <ul style="list-style-type: none"> None: Nothing is done DeleteFile: The file is deleted on rotation RenameFileInTheSameDirectory: The file is renamed as per the onrotationoptions parameter, described below on rotation.
agents[x].onrotationoptions	processed	<p>If the onrotation parameter is chosen as "RenameFileInTheSameDirectory", this parameter tells what to rename the file.</p> <p>For example: If the default value is processed, on rotation, the file sample.log is renamed to sample.log.processed.</p> <p>The Unix extension cannot have spaces in it.</p>
agents[x].preservestate	false	<p>If set to true, remembers the last location read in the file periodically, depending on the values set for the perservedstatecount and preservedstateinterval properties.</p> <p>If set to false, then nothing is written and the connector has no record of where it left off. In this case, the values of perservedstatecount and preservedstateinterval are ignored.</p>
agents[x].preservedstatecount	10	The number of times the value has to change or has to be updated before actually preserving the state.
agents[x].preservedstateinterval	30000	The number of idle milliseconds that will trigger a state persistence.
agents[x].rotationsleeptime	10	Used in conjunction with rotationonlywheneventexists, rotation will not occur until the specified time has elapsed since the new event appeared. Default is 10 seconds.
agents[x].rotationscheme	Daily	Possible values: Daily, Index, None
agents[x].rotationonlywheneventexists	false	Used only by the daily log follower in conjunction with rotationsleeptime, no rotation occurs until there is new event in the file or there is a new event and the time for rotationsleeptime has elapsed since the new event appeared. Default is false - not enabled.
agents[x].rotationschemeparams	[blank]	<p>Configure this parameter when rotationscheme parameter is set to Daily or Index.</p> <p>A filename template has the following syntax:</p> <p><i>prefix</i>,<i>dateFormat</i>,<i>suffix</i>[true false]</p> <p>For a complete description of how to use the rotationschemeparams parameter, see "Parameters for Daily and Index Rotation"</p>
agents[x].rotationdelay	30	In seconds. Specifies how long to wait after a new file is detected before the file reader thread for the current file is terminated and a file reader thread launches for a new file.

Parameter	Default	Description
agents[x].startatend	true	The default is true. Useful when log files to be processed already exist and contain data at connector startup or when the log file rotation takes place. Setting this value to false will cause the entire file to be read at every startup, which could lead to duplicate events, unless the preservestate parameter is set to true. (Setting preservestate to true lets the connector skip the old events and start from the last preserved read position of the file.)
agents[x].usealternate rotationdetection	true	Use an alternate mechanism to detect log rotation. Used with followexternalrotation parameter. The log rotation detection logic uses a file's length as opposed the number of bytes counted in byte counting input stream. Setting this value to true, compares a new file length to the previous file length. Setting the value to false, compares a new file length to the number of bytes read from the file (input stream).
agents[x].usenonlocking windowsfilereader	true	Does not lock the log file read by the connector on the Windows platform.

File Folder Follower Parameters

The following table describes the File Folder Follower parameters. If you do not see the parameter you need in the table, see "[File Connector Parameters](#)".

Parameter	Default	Description
agents[x].delay	10000	In milliseconds. Specifies how long the connector waits to start before processing after it detects the file for the first time in the folder.
agents[x].encoding	UTF8	Specifies the encoding or character set used in the log file. Only Java recognized encoding is accepted. Informal names for encoding will result in assuming UTF8 as logs encoding value.
agents[x].fixedlinelength Note: For SAP only	-1	If set to a positive integer, this parameter sets the line length for an event. The length can be expressed as either the number of characters or bytes. The -1 default value indicates that one line represents one event. This is because one line is typically one event.
agents[x].fixedlinelength contains Note: For SAP only	[Fixed Number of Characters.]	Related to the fixedlinelength parameter. Specifies whether the fixed length is the number of bytes or number of characters. Possible Values are Fixed Number Of Bytes (default) or Fixed Number Of Characters.

Parameter	Default	Description
agents[x].followexternal rotation	false	<p>Specifies whether the file reader thread is going to follow any rotation to the file done by the external device writing to the log. Is operational only when agents[x].processingmode is set to realtime.</p> <p>If property is set to false, the rotation “in place” is not followed, the file read once, the drop in size may not be monitored. If set to true, it will be monitored. There are different ways files are rotated:</p> <ul style="list-style-type: none"> • if the new file has “index” suffix, for example, log1.txt, log2.txt • if the new file has a new date stamp added, for example, log2013-07-31.txt for daily rotation <p>These types of rotation are captured with “rotationscheme” and related parameters, or some rotation specified directly in file name regex, like those based on multifolderfollower (Apache Tomcat is an example).</p> <p>If this property is set to true, the Connector will monitor the size of the file (using the file’s name, not its inode). If the file size has decreased, the Connector will assume that the file has been rotated.</p>
agents[x].maxretries	-1	<p>Maximum number of retries before giving up on a file.</p> <p>Files will be moved to the “bad” directory if unable to read at once. Positive means retry up to maxretries times to read again.</p>
agents[x].minfilelength	-1	Prevents processing of files smaller than the specified size.
agents[x].mode	RenameFile InTheSameDirectory	<p>Specifies the action to perform on a log file after the Connector has processed it. Possible actions are:</p> <ul style="list-style-type: none"> • RenameFileInTheSameDirectory—Renames the processed log file to filename.processed. • DeleteFile—Deletes the file once it has been processed. • PersistFile—Retains the file with its original name after it has been processed. However, the Connector remembers the files it has already processed so that those are not processed again. <p>Note: The value for agents[x].usenonlockingwindowsfilereader must be set to true in Windows environments for the modes RenameFileInTheSameDirectory and DeleteFile to function correctly.</p>
agents[x].modeoptions	processed	<p>Specifies the extension to add to processed files.</p> <p>For example, .processed.</p> <p>The Unix extension cannot have spaces in it.</p>

Parameter	Default	Description
agents[x].monitoringinterval	30000	<p>Specifies the amount of time (in milliseconds) that the connector will wait before re-reading the log file. The connector checks if file was updated; if it was, then the connector continues to read the file.</p> <p>After the file is read to the EOF, the connector checks for new records until the value of the processingtimeout parameter is reached. If no updates have occurred, then the connector checks for updates only at intervals equal to the value of the monitoringinterval parameter. If no updates have occurred up to the value of the processingthreshold parameter, then the connector marks file as done and terminates reading.</p> <p>The monitoringinterval parameter should be used only when the processingmode parameter is set to realtime. The value of the monitoringinterval parameter must be greater than 0 and less than the value of the processing timeout parameter ($0 < \text{monitoringinterval} < \text{processingtimeout} < \text{processingthreshold}$).</p>
agents[x].processfolder recursively	false	<p>Specifies whether to process log files in the subfolders of a specified folder.</p> <p>Possible values: true and false</p> <p>When this property is set to true, the Connector traverses the subfolders in a folder to locate log files to process.</p>
agents[x].processinglimit	256	<p>Set to specify the number of files to read in real time. There is one file reader thread per file. When this limit is reached no new files will be processed until some of the existing files are temporarily suspended because of inactivity or are completely processed.</p>
agents[x].processingmode	batch	<p>Specifies the mode for connector log file processing. Possible values are:</p> <ul style="list-style-type: none"> batch—Batch processing of the log file. realtime—Realtime processing of the log file. <p>If realtime is specified, then the properties, monitoringinterval, processingthreshold, and processingtimeout must also be specified.</p>

Parameter	Default	Description
agents[x].processingthreshold	3600000	<p>Specifies the amount of time (in milliseconds) that the connector will wait for inactivity on the realtime log file. When the processingthreshold value is exceeded, the log file is deleted or renamed depending on the agents[x].mode value specified.</p> <p>The processingthreshold parameter should be used only when the processingmode parameter is set to realtime. The value of the processingthreshold parameter must be greater than 0 and greater than the value of the processingtimeout parameter ($0 < \text{monitoringinterval} < \text{processingtimeoutparameter} < \text{processingthreshold}$).</p> <p>This parameter cannot be disabled. Therefore, when processing mode is set to 'realtime' and 'followexternalrotation' is set to 'true', the connector may stop reading from the file over time. Although the default value is 3600000, you can specify a larger value. The max value is 9223372036854775807 (292,471,208.67753601074 years).</p> <div> Note: When the processingthreshold value is exceeded, the log file is deleted or renamed depending on the agents[x].mode value specified. </div>
agents[x].processingtimeout	120000	<p>Specifies the threshold time (in milliseconds) for detecting inactivity on the realtime log file. If inactivity on the log file exceeds this value, then reading of the log file is suspended. The log file is again checked whether to suspend, resume or terminate after the monitoringinterval has elapsed.</p> <p>The processingtimeout parameter should be used only when the processingmode parameter is set to realtime. The value of the processingtimeout parameter must be greater than 0 and less than the value of the processingthreshold parameter ($0 < \text{monitoringinterval} < \text{processingtimeout} < \text{processingthreshold}$).</p>
agents[x].retryinterval	1000	In milliseconds. If you want to try again to process unprocessed files (which were not processed because of an exception, such as a busy device), use these fields.
agents[x].sleeptime	5000	Specified how long to wait before checking the folder for new files.
agents[x].triggerextension	.done	<p>This parameter is used only if usertriggerfile parameter is set to true.</p> <p>Specifies the file extension that the connector should look for to identify a trigger file. Used in conjunction with usetriggerfile. It can be any word at the end.</p> <p>For example, .trigger.</p>

Parameter	Default	Description
agents [x].usealternaterotation detection	false	Use an alternate mechanism to detect log rotation. Used with followexternalrotation parameter. It tells the log rotation detection logic to use a file's length as opposed the number of bytes counted in byte counting input stream.
agents[x].usenonlocking windowsfilereader	false	Does not lock the log file read by the connector on the Windows platform so the device writing the log can rotate it if it chooses. On Windows platform one process that writes into the file can prevent the other process from reading. "true" allows the connector to read the file regardless. Connector never locks the file, it is always only a reader.
agents[x].usetriggerfile	false	Specifies whether to look for a trigger file before processing a log file. Possible values: true or false A trigger file is an empty file that has the same name as the log file, but a different extension. This file is created by certain systems to indicate that a log file is ready for processing. If this property is set to true, the connector will not process a log file until a trigger file for it has been created in the same folder where the log file exists.
agents[x].wildcard	[blank]	Use the wildcard parameter to match file names for daily or index file rotation. The wildcard parameter can be used only for file folder follower connectors and has special restrictions for Regex File connectors. For more information on how to use the wildcard parameter and its syntax, see "Using wildcard for Daily and Index Log File Rotation (File Folder Follower Only)" .

Syslog Parameters

This section contains information on the following Syslog parameters:

- [Syslog Daemon Parameters](#)
- [Syslog Pipe Parameters](#)
- [Syslog File Parameters](#)
- [Syslog NG Daemon Parameters](#)
- [Raw Syslog Daemon Parameters](#)
- [ArcSight CEF Encrypted Syslog \(UDP\) Parameters](#)
- [TippingPoint SMS Syslog Extended Parameters](#)

Syslog Daemon Parameters

The following sections describe the Syslog Daemon parameters.

- [Event Parsing \(Sub-agents\) Parameters](#)
- [Event Reception Parameters](#)
- [Raw Log Parameters](#)
- [Event Queue Parameters](#)
- [Event Processing Parameters](#)

Event Parsing (Sub-agents) Parameters

The following table describes the Syslog Event Parsing (Sub-agents) parameters.

Parameter	Default	Description
agents[x].customsubagentlist	[The default is too long to display.]	<p>Set this property to the restricted subagent list based on device types in your environment. List parsers' names separated by (vertical bar) and no " " (quotes) are allowed.</p> <p>This parameter is used in conjunction with agents [x].usecustomsubagentlist. It can help reduce the time the connector needs to pick up the right parser.</p> <p>Examples: agents[x].customsubagentlist=ciscopix_syslog if your Connector is designed to parse cisco pix syslog events. Or agents[x].customsubagentlist=ciscopix_syslog cyberguard_syslog if your Connectors are going to take care of those 2 kinds of events.</p>
agents[x].forwardmode	false	<p>If set to true, every message is run through every available syslog parser and the first parser whose main regex matches the message is assumed to be the correct parser. This is very inefficient because every message potentially must be run through 100+ parsers. However, in this mode, the message is less likely to be picked up by the incorrect parser.</p> <p>If set to false, the connector will pick the first match parser and save time. However, this mode may raise the chance that the wrong parser is picked.</p>
agents[x].usecustomsubagentlist	false	<p>Set to true to use the agents[x].customsubagentlist property. This makes the connector to consider the customized subagent list.</p>

Event Reception Parameters

The following table describes the Syslog Event Reception parameters.

Parameter	Default	Description
agents[x].encoding	[blank]	<p>By default, there is no entry for agents[x].encoding in agent property file. If you want to use an alternative value, add this parameter manually.</p> <p>If this is specified and is valid, the specified encoding (for example, UTF-16) is used. If not, then the default depends on the protocol: UTF-8 for Raw TCP or the platform default for UDP. UTF-16 would be an example of a value to set the property.</p>
agents[x].tcpbindretrytime	5000	<p>Time between TCP bind retries (in milliseconds). Time gap to retry to bind to a socket address.</p>
agents[x].tcpbuffersize	10240	<p>Raw TCP buffer (in bytes). This is the initial size. It will be expanded if necessary, up to the value defined by tcpmaxbuffersize parameter.</p> <p>By default, tcpbuffersize is 10k and tcpmaxbuffersize is 1M. The reason we set tcpbuffersize small is to save system resources.</p> <p>Example of how these two parameters work with each other:</p> <p>When a single tcp event is less than 10k in length (most of the event won't be longer than this), nothing will be changed.</p> <p>When a single tcp event is 15k in length, the connector will first try with 10k tcpbuffersize and if failed, it will check if tcpbuffersize exceeded the limit of tcpmaxbuffersize. If not, it will double tcpbuffersize to 20k, and find if 20k buffer is efficient to hold the 15k tcp event. After this event, the system will continue with the 20k tcpbuffersize and 1M tcpmaxbuffersize.</p> <p>When the single tcp event is 25k in length and the connector finds the 20k tcpbuffersize is still not enough to hold the event, it will double tcpbuffersize again until it can hold the event. After this event, the system will continue to work with 40k tcpbuffersize.</p> <p>But when the tcp event is even larger than tcpmaxbuffersize, for example 1.5M (this is rare), the connector keeps doubling its tcpbuffersize until it reaches tcpmaxbuffersize. And then it will truncate the 1.5M event immediately.</p> <p>Notice you can also set tcpmaxbuffersize < tcpmaxbuffersize = tcpbuffersize > at the beginning. The connector will still use tcpbuffersize to measure the event length. When the event is longer than tcpbuffersize, it won't try to expand; instead, it will truncate the event immediately.</p>

Parameter	Default	Description
agents[x].tcpcleanupdelay	-1	<p>Idle TCP cleanup delay (in milliseconds). How often the idle TCP socket should be cleaned up. The default value of -1 indicates the idle TCP socket is never cleaned up.</p> <p>Note that both tcpcleanupdelay and tcpmaxidletime must be set to values greater than zero in order for idle TCP sockets to be cleaned up. Also, if tcppeerclosedchecktimeout is set it takes precedence.</p>
agents[x].tcpendchar	[blank]	<p>Optional message terminating hex character, can use either 0x00 or NUL. Not defined by default.</p>
agents[x].tcpmaxsockets	1000	<p>Specifies the maximum number of TCP connections that connector will accept simultaneously. Parameter value should be any positive integer that can reasonably utilize system resource and won't crash the system. Connector will only accept TCP connection when the total connection is under the number defined by this parameter, as soon as exceeded, the connection will be rejected and print out fatal message. The default value of this parameter is 1000, increase this value as required to accommodate simultaneous connections from a large number of devices.</p>
agents[x].tcpmaxbufferize	1 MB	<p>Maximum raw TCP buffer (in bytes). Any message larger than the given size will be truncated. The tcpmaxbufferize is not used to truncate events, this value is used to limit the expansion of tcpbufferize, and how to cut event is determined by tcpbufferize. See also agents[x].tcpbufferize.</p>
agents[x].tcpsleeptime	50	<p>If no data because either no sockets or sockets have no data, then sleep this long (in milliseconds) before checking again.</p>
agents[x].overwriterawevent	false	<p>With the default value of false, if the parser for this syslog device directly sets the rawEvent event field, the connector leaves that value as is. And if the parser does not set that event field, then the full syslog message is put into that event field. If this property is changed to true, then the full syslog message is always used, even if that means overwriting a rawEvent value that was explicitly set by the parser.</p>

Raw Log Parameters

The following table describes the Syslog Raw Log parameters.

Parameter	Default	Description
agents[x].rawlogfolder	[blank]	<p>This parameter defines the folder used for storing raw logs files.</p> <p>By default this parameter is omitted from the agent.properties file. If you want to store raw log files add this parameter and specify the folder where the raw log files are stored. The connector creates the specified folder if it does not exist.</p> <p>The value for this property can be an absolute path for a folder in which to store the raw log files. For example:</p> <p>agents[x].rawlogfolder=/opt/arcsight</p> <p>will cause the raw log files to be stored in the /opt/arcsight folder.</p> <p>Alternatively the value for this property can be a relative path. This path is prefixed with \$ARCSIGHT_HOME/current/user/agent/ to form the full path. For example:</p> <p>agents[x].rawlogfolder=arcsight</p> <p>will cause the raw log files to be stored in the \$ARCSIGHT_HOME/current/user/agent/arcsight folder.</p> <p>If value given for rawlogfolder contains any invalid character for path and folder name (e.g. '<' and '>' on Windows), then the raw log files will be stored in \$ARCSIGHT_HOME/current/user/agent/agentdata.</p> <p>If you want to use the rawlogfolder feature, then you must set this parameter and also set at least one of agents[x].rawloginterval and agents[x].rawlogmaxsize to positive values.</p>
agents[x].rawloginterval	-1	<p>The raw log interval before each rotation (in seconds, or -1 to not rotate based on time), if rawlogfolder is enabled.</p>
agents[x].rawlogmaxsize	-1	<p>Raw event log maximum size in MB, or -1 to not rotate on size, if rawlogfolder is enabled.</p> <p>If both rawloginterval and rawlogmaxsize have positive values then both values are used to control log rotation. Log rotation occurs whenever either of the values are reached.</p>

Event Queue Parameters

The following table describes the Syslog Event Queue parameters.

Parameter	Default	Description
agents[x].filequeuemaxfilecount	100	File queue maximum file count. If the number passes filequeuemaxfilecount, the connector starts to take action to avoid filling up the disk. An action can be dropping events or whole files may be omitted. It is important to choose the filequeuemaxfilecount value carefully to avoid losing data.
agents[x].filequeuemaxfilesize	10000000	File queue max file size (Bytes). Increase this parameter to increase the size of each file in the file queue.
agents[x].usefilequeue	true	<p>This parameter is to determine whether Connector will keep the raw events received into a file queue consisting of a certain number of fix-sized files.</p> <p>Possible values: true and false</p> <p>When this parameter is set to true the connector stores raw events in files as they are received and then processes the events by reading the files. Using file queues helps avoid event loss when bursts of events are arrive faster than they can be processed. The values for filequeuemaxfilecount and filequeuemaxfilesize are used to define the file queue behavior.</p> <p>When this parameter is set to false file queues are not used and the values for filequeuemaxfilecount and filequeuemaxfilesize are ignored.</p>

Note: usefilequeue and all related properties cannot be applied to Syslog Pipe/File connectors.

Event Processing Parameters

The following table describes the Syslog Event Processing parameters.

Parameter	Default	Description
agents[x].aggregationcachesize	1000	<p>Aggregation Cache Size.</p> <p>For syslog connectors the aggregation cache stores the last event received from each distinct source. When an event is received that indicates “last message repeated n times” the stored event is used as the security event, marked as “aggregated” and annotated with the repetition count.</p> <p>Parameter aggregationcachesize specifies the maximum number of aggregation cache entries. Avoid configurations where there are more than aggregationcachesize sources. Aggregation is not done for a source whose event is not stored in the cache due to the aggregationcachesize being exceeded.</p>
syslog.setdevicehostname conservatively.syslog	false	<p>With the default value of false, the deviceHostName event field is set based on where the syslog message came from, before the parser operates. If this property is changed to true, then the deviceHostName event field is similarly set but after the parser operates, and only if neither the deviceHostName nor the deviceAddress event fields were set by the parser.</p> <p>This is a container level parameter.</p>

Syslog Pipe Parameters

The following table describes the Syslog Pipe parameters. All of the parameters described under "[Syslog Daemon Parameters](#)" also apply to Syslog Pipe.

Parameter	Default	Description
agents[x].configrestartsleepime	5000	Time (in milliseconds) to wait before sending the configuration restart signal to Syslog when running on Solaris.
agents[x].sleepime	5	Time (in seconds) to wait between file polling after pipe has ended.
agents[x].solarissyslogconfigrestart command	kill -HUP 'cat/etc/syslog.pid'	Configuration restart signal to Syslog/Command to execute after the connector starts reading the pipe when running on Solaris.
syslog.setdevicehostname conservatively.syslog_pipe	false	With the default value of false, the deviceHostName event field is set based on where the syslog message came from, before the parser operates. If this property is changed to true, then the deviceHostName event field is similarly set but after the parser operates, and only if neither the deviceHostName nor the deviceAddress event fields were set by the parser. This is a container level parameter.

Syslog File Parameters

The following table describes the Syslog File parameters. All the parameters described under "[Syslog Daemon Parameters](#)" apply.

Parameter	Default	Description
agents [x].internalevent.filestart.enable	true	When true, an internal audit event is generated whenever the connector opens a file for processing. If you don't want to receive ArcSight internal events, you can turn this off.
agents[x].internalevent.fileend.enable	true	When true, an internal audit event is generated whenever the connector completes processing a file.

Parameter	Default	Description
agents[x].internalevent.filecount.enable	false	<p>This feature has the following parameters:</p> <p>agents[x].internalevent.filecount.duration=nnn</p> <p>Specifies the number of seconds that the connector has to process a specified number of files.</p> <p>agents[x].internalevent.filecount.minfilecount=nnn</p> <p>Specifies the minimum number of files that the connector should process in a specified number of seconds.</p> <p>agents[x].internalevent.filecount.timer.delay=nnn</p> <p>Specifies, in seconds, how often the connector should check to see if the connector is compliant with the other parameters.</p> <p>If agents[x].internalevent.filecount.enable=true and all three affiliate parameters are set appropriately, every internalevent.filecount.timer.delay second, the connector checks if in the last internalevent.filecount.duration second, the connector processed enough events defined by the internalevent.filecount.minfilecount.</p> <p>If not, the connector will send an internal event to the destination with the name:</p> <p><i>Number of files processed is less than expected value.</i></p>
agents[x].startatend	true	<p>If set to true, the connector receives only new lines inserted into the file.</p> <p>If set to false, when the connector starts to process a file, it will process the whole file.</p>
syslog.setdevicehostname conservatively.syslog_file	false	<p>With the default value of false, the deviceHostName event field is set based on where the syslog message came from, before the parser operates. If this property is changed to true, then the deviceHostName event field is similarly set but after the parser operates, and only if neither the deviceHostName nor the deviceAddress event fields were set by the parser.</p> <p>This is a container level parameter.</p>

Syslog NG Daemon Parameters

The following table describes the Syslog NG Daemon parameters. All the parameters described under "[Syslog Daemon Parameters](#)" apply.

Parameter	Default	Description
agents[x].syslogng.mutual.auth.enabled	false (disabled)	Determines whether mutual authentication is enabled for TLS. If false, mutual authentication is disabled, and the Syslog NG agent is authenticated by the client. If true, mutual authentication is enabled, and the client is authenticated by the Syslog NG agent.
agents[x].syslogng.subagents.with.ietf	generic_syslog	List of subagents for SyslogNG (when IETF format is enabled).
syslogng.tls.cert.file	user/agent/syslog-ng.cert	Location for the cert file to be used by Syslog NG clients for TLS communication with the Syslog NG agent. This is a container level parameter.
syslogng.header	(?s)^(?:\\d{1})?\\s+(\\S+)\\s+(\\S+)\\s+(*)	Pattern to parse the header and extract out SYSLOG-VERSION, TIMESTAMP, HOST, REST_OF_MESSAGE This is a container level parameter.
syslogng.header.tag	(?s)^(\\S+)\\s+(\\S+)\\s+(\\S+)\\s+(-(?:\\[\\S+@[^\\]]+\\])+\\s+(*))	Pattern to parse the header and extract out APPNAME, PROCID, MSGID, STRUCTURED_DATA, MESSAGE These are specific parameters for syslog input format. They can be changed if needed, but it is a rare occurrence. This is a container level parameter.
syslogng.header.timestamp	(?s)^(\\d{4}-\\d{2}-\\d{2}T\\d{2}:\\d{2}:\\d{2})(X \\d+)?(Z (?-\\ +))\\d{2}:\\d{2}?	Pattern to parse the time stamp. Example of a time stamp string that could be parsed by the default pattern: 1985-04-12T23:20:50.52Z This is a container level parameter.

Raw Syslog Daemon Parameters

The following table describes the Raw Syslog Daemon parameters. All of the parameters described under "[Syslog Daemon Parameters](#)" and "[Syslog NG Daemon Parameters](#)" apply.

Parameter	Default	Description
agents[x].simpletimestampformat	[Blank]	Custom format for Captured Timestamp. This uses Java's SimpleDateFormat pattern syntax. If left blank, the <code>_parseMutableTimeStampSilently</code> operation is used to parse the time stamp.

ArcSight CEF Encrypted Syslog (UDP) Parameters

The following table describes the ArcSight CEF Encrypted Syslog (UDP) parameters. All of the parameters described under "[Syslog Daemon Parameters](#)" apply.

Parameter	Default	Description
agents[x].customsubagentlist	cef_syslog	cef_syslog is the only supported value for this parameter. Do not change the default for this property.
agents[x].protocol	Encrypted UDP	Encrypted UDP is the only supported value for this parameter. Do not change the default for this property.
agents[x].usecustomsubagentlist	true	Indicates whether this agent uses the custom subagent list (cef_syslog). Do not change the default for this property.

TippingPoint SMS Syslog Extended Parameters

The following table describes the TippingPoint SMS Syslog Extended parameters. All of the parameters described under "[Syslog Daemon Parameters](#)" apply.

Parameter	Default	Description
agents[x].eventidfilepath	ARCSIGHT_HOME/user/agent/	Path for event ID file.
agents[x].syslogmode	SyslogD	Syslog Mode can be SyslogD, Pipe or File. Only SyslogD is supported. Do not change this value.
syslog.setdevicehostnameconservatively. tippingpoint_sms_syslog	false	With the default value of false, the deviceHostName event field is set based on where the syslog message came from, before the parser operates. If this property is changed to true, then the deviceHostName event field is similarly set but after the parser operates, and only if neither the deviceHostName nor the deviceAddress event fields were set by the parser. This is a container level parameter.

Appendix H: FlexConnectors and Categorization

This topic describes categorization in the context of FlexConnectors.

For details on using categorization, see the *ArcSight Console User's Guide*, Reference Guide section, specifically the topic “Event Categorization” for information on custom categorization for FlexConnectors.

Categorization

You can categorize the events collected by your FlexConnector. The following examples illustrate categorization for HTTP status code-based devices (such as proxy, cache, or web servers) and for Firewall devices (which use pass/open/allow, drop/deny/reject). Put the categorization file in this location:

```
ARCSIGHT_HOME/user/agent/acp/categorizer/current/  
<device_vendor>/  
<device_product>.csv
```

In this case, **<device_vendor>** is the value of the **event.deviceVendor** field (in lower case and with spaces or other special characters replaced by an underline). The **<device_product>** is the value the **event.deviceProduct** field (likewise in lower case with spaces replaced by underlines). Your FlexConnector must set these fields before you can use categorization.

HTTP Status Code Categorization Example

```
event.deviceEventClassId,set.event.categoryObject,  
set.event.categoryBehavior,set.event.categoryTechnique,set.event.  
categoryDeviceGroup,set.event.categorySignificance,set.event.  
categoryOutcome  
100,/Host/Application/Service,/Communicate/Query,,/Application,/Informational,/Success  
101,/Host/Application/Service,/Communicate/Query,,/Application,/Informational/Error,/Attempt  
200,/Host/Application/Service,/Communicate/Query,,/Application,/Normal,/Success  
201,/Host/Resource,/Create,,/Application,/Normal,/Success  
202,/Host/Application,/Execute,,/Application,/Informational/Error,/Failure  
203,,,,/Application,,  
204,/Host/Resource,/Access/Start,,/Application,/Normal,/Success  
205,/Host/Resource,/Access/Start,,/Application,/Informational,/
```

Success

206,/Host/Resource,/Access/Start,,/Application,/Informational,/Success
300,/Host/Resource,/Access/Start,,/Application,/Informational,/Success
301,/Host/Application/Service,/Communicate/Query,/Redirection/Application,/Application,/Informational,/Success
302,/Host/Application/Service,/Communicate/Query,/Redirection/Application,/Application,/Informational,/Success
303,/Host/Application/Service,/Communicate/Query,/Redirection/Application,/Application,/Informational,/Success
304,/Host/Application/Service,/Communicate/Query,/Redirection/Application,/Application,/Informational,/Success
305,/Host/Application/Service,/Communicate/Query,/Redirection/Application,/Application,/Informational/Error,/Attempt
306,/Host/Application/Service,/Execute/Query,,/Application,/Informational/Alert,/Failure
307,/Host/Application/Service,/Communicate/Query,/Redirection/Application,/Application,/Informational,/Success
400,/Host/Application/Service,/Access/Start,/Traffic Anomaly/Application Layer/Syntax Error,/Application,/Informational/Warning,/Failure
401,/Host/Application/Service,/Authentication/Verify,,/Application,/Informational/Warning,/Failure
402,/Host/Application/Service,/Communicate/Query,/Traffic Anomaly/Application Layer/Unsupported Command,/Application,/Informational/Error,/Failure
403,/Host/Application/Service,/Authentication/Verify,,/Application,/Informational/Warning,/Failure
404,/Host/Resource,/Access/Start,,/Application,/Informational/Warning,/Failure
405,/Host/Application/Service,/Communicate/Query,/Traffic Anomaly/Application Layer/Unsupported Command,/Application,/Informational/Error,/Failure
406,/Host/Application/Service,/Communicate/Query,,/Application,/Informational/Error,/Failure
407,/Host/Application/Service,/Authentication,,/Application,/Informational/Error,/Failure
408,/Host/Application/Service,/Communicate/Query,,/Application,/Informational/Error,/Failure
409,/Host/Application/Service,/Communicate/Query,,/Application,/

Informational/Error,/Failure
410,/Host/Resource,/Access/Start,,/Application,/Informational/
Warning,/Failure
411,/Host/Application/Service,/Access/Start,/Traffic
Anomaly/Application Layer/Syntax
Error,/Application,/Informational/Warning,/Failure
412,/Host/Application/Service,/Access/Start,,/Application,/Informational/Warning,/Failure
413,/Host/Application/Service,/Communicate/Query,/Traffic Anomaly/Application Layer/Syntax
Error,/Application,/Informational/Error,/Failure
414,/Host/Application/Service,/Communicate/Query,/Traffic Anomaly/Application Layer/Syntax
Error,/Application,/Informational/Error,/Failure
415,/Host/Application/Service,/Communicate/Query,/Traffic Anomaly/Application Layer/Syntax
Error,/Application,/Informational/Error,/Failure
416,/Host/Application/Service,/Communicate/Query,/Traffic Anomaly/Application Layer/Syntax
Error,/Application,/Informational/Error,/Failure
417,/Host/Application/Service,/Communicate/Query,/Traffic Anomaly/Application Layer/Syntax
Error,/Application,/Informational/Error,/Failure
500,/Host/Application/Service,/Execute,,/Application,/Informational/Error,/Failure
501,/Host/Application/Service,/Execute,,/Application,/Informational/Error,/Failure
502,/Host/Application/Service,/Execute,,/Application,/Informational/Error,/Failure
503,/Host/Application/Service,/Access/Start,,/Application,/Informational/Error,/Failure
504,/Host/Application/Service,/Execute,,/Application,/Informational/Error,/Failure

Firewall Example

```
event.deviceEventClassId,set.event.categoryObject,  
set.event.categoryBehavior,set.event.categoryDeviceGroup,  
set.event.categorySignificance,set.event.categoryOutcome  
OPEN,/Host/Application/Service,/Communicate/Query,/Firewall,/Normal,/Success  
pass,/Host/Application/Service,/Communicate/Query,/Firewall,/Normal,/Success  
DROP,/Host/Application/Service,/Communicate/Query,/Firewall,/Informational/Warning,/Failure
```


Appendix I: Developing a Syslog FlexConnector

Follow these general steps to create a syslog FlexConnector.

1. Capture the RAW syslog by sending samples to a receiver (that is, the machine where the connector is installed). Capture the logs by enabling RAW event. Extract the ASCII and modify the logs to represent syslog format. Analyze the raw syslog.

Note: Do not pull the RAW syslog from the logger, because it does not format the output correctly.

2. Save a small sample of the file and use it as input to the ArcSight Regex Tool (`/($HOME/user/current/bin/arcsight.bat regex)`). See "[Regex Tool for Regex FlexConnectors](#)".

The Regex Tool will automatically detect the syslog header if it is in the correct format (that is, **timestamp hostname/hostIP**). Otherwise, it will not.

3. Choose syslog subagent in the Regex Tool (**Options > Treat as Syslog SubAgent**). The regex should be able to start after the automatically detected syslog header that the Regex Tool picks up.
4. Name the file as **myCompany_syslog.subagent.sdkfilereader.properties**.
5. Complete the coding and mapping. For a list of the syslog tokens that can be used in the parser for mapping to event fields, see "[ArcSight Built-in Tokens](#)". Save the **.properties** file in the **/current/user/agent/flexagent/syslog** folder on the syslog receiver you are using.
6. Verify the value of the **agents[0].usecustomsubagentlist** property is set to **true** in the **agent.properties** file:

agents[0].usecustomsubagentlist=true

7. Add an entry in **agent.properties** file to include your custom subagent name. The **agent.properties** file already contains a list of subagents. Ensure that your FlexConnector subagent name is first in the list, for example:

agents[0].customsubagentlist=MyCompany_syslog

...

8. Test the syslog connector against the new parser.

The connector automatically generates an entry in the **syslog.properties** file when syslog is running. This value should be automatically detected by the FlexConnector. Verify that the property is present in the file, for example:

...

```
syslog.subagentdef=xxx.xxx.x.xx\:myCompany_syslog|flexagent_syslog|generic_  
syslog
```

Appendix J: Developing an XML FlexConnector

You can create an XML FlexConnector to recursively read events from XML-based files in a folder. Choose the XML FlexConnector for devices that write event information to XML files, such as vulnerability scanners that produce XML reports.

The following topics are covered:

- [XML FlexConnector Development](#)
- [XML Tools](#)
- [XML Concepts for FlexConnector Development](#)
- [Prepare to Write the Parser - Identify Namespace, Nodes, and Tokens](#)
- [Create the XML FlexConnector Parser](#)
- [Install the FlexConnector](#)

XML FlexConnector Development

Use XML tools to read the XML log files that you are using as your source for your parser. The sections below breakdown parser development, categorization, and XML FlexConnector installation.

XML Tools

You can use various XML query tools to edit XML documents, find information in XML documents, or to extract elements and attributes from XML documents to use in parser creation.

XML query tools include XPath and XQuery, which are available from:

- <http://www.w3schools.com/xpath> (XPath is a language for finding information in XML documents)
- <http://www.w3schools.com/xquery> (XQuery is a tool for finding and extracting elements and attributes from XML documents)

These pages contain additional information on using XQuery:

- http://www.stylusstudio.com/xquery_primer.html
- http://www.stylusstudio.com/xquery_flwor.html
- <http://www.xqueryfunctions.com/xq/alpha.html> (XQuery function library; useful for building expressions)

These are some XML editors:

- <http://www.mindfusion.eu/product1.html> (XML Viewer)
- http://www.stylusstudio.com/xml_download.html (Stylus Studio XML)
- <http://www.altova.com/download-trial.html> (Altova XML Spy)

Tools like these are useful for parser creation. Try these or you might find other similar tools on the web that you like better.

XML Concepts for FlexConnector Development

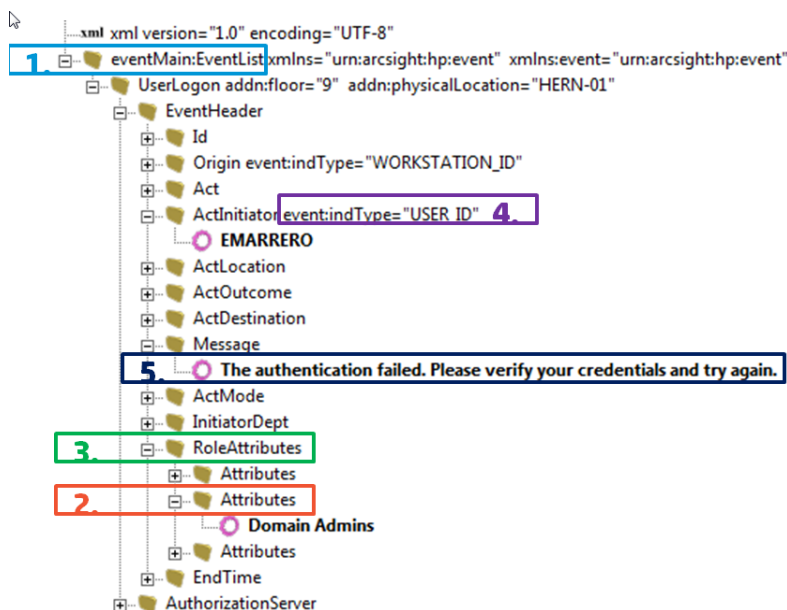
These are some useful XML concepts that will help you develop your XML FlexConnector.

General XML Concepts

These are some concepts that are common to XML files, but that are good for you to keep in mind when you are creating your parser:

The following example of an XML file is annotated to highlight the code that corresponds with these key concepts:

1. Root Node
2. Leaf Nodes
3. Intermediate Nodes
4. Attributes
5. Text



XML FlexConnector Concepts

These are some concepts that are specific to XML FlexConnector parsers:

- [Namespace](#)
- [Hop Nodes](#)
- [Trigger Nodes](#)
- [Token Mappings](#)
- [Extra Events](#)

Namespace

Use if your XML log file uses explicit namespaces or a default namespace in the header. Using namespaces allows you to differentiate between elements that have the same name in the schema, but actually refer to different content.

Specify those namespaces using these properties:

- **namespace.count**—Specifies the number of namespaces that your XML log file uses; for example, **namespace.count=2**.
- **namespace.prefix**—Specifies the namespace prefix to use; for example, **namespace[1].prefix=ac**.
- **namespace[x].prefix=default**—Use when your XML file specifies a namespace but does not use any prefixes in the file. That is, your XML file uses a default namespace.
- **namespace.uri**—Specifies the Uniform Resource Identifier (URI) for the namespace; for example, **namespace[0].uri=http://example.org/2003/08/sdee**

For example:

```
namespace.count=2
```

```
namespace[0].prefix=default  
namespace[0].uri=http://www.mycompany.com/ids/2014/09/example
```

```
namespace[1].prefix=ac  
namespace[1].uri=http://www.yourcompany.com/fds/acfg
```

Hop Nodes

Optional. Hop nodes are the nodes in the path from the root node to the event triggering node. These nodes are necessary when tokens need to be captured from nodes other than the triggering node or when events pertaining to a particular node need to be grouped in one block. Select nodes other than the trigger node that contain relevant security event information to be hop nodes.

Multiple hop node levels can be defined with each new level of hop nodes defined in reference to the previously defined level. Hop nodes can also reference root nodes directly as variables.

To define hop nodes, use these properties:

- **hop.node.count**—Specifies the number of hop nodes; for example, **hop.node.count=1**
- **hop.node.name**—Specifies the names for the hop nodes; for example, **hop.node[0].name=host**
- **hop.node.expression**—Specifies the XPath/XQuery path expressions to select the nodes; for example, **hop.node[1].expression=/audits/audit/hosts/host**

For example:

```
hop.node.count=1
hop.node[0].name=host
hop.node[0].expression=/audits/audit/hosts/host
```

Trigger Nodes

Mandatory. These are the nodes that trigger events. An XPath/XQuery path expression for a trigger node can be the last defined hop node or the root node if no hop nodes are available.

The number of trigger nodes determines the number of events that are generated using your parser. The parser will generate an event each time the trigger node is discovered in the log file.

To define trigger nodes, use a property like this:

```
trigger.node.expression=$host/applications/application,$host/
vulnerabilities/vulnerability
```

Token Mappings

Mandatory. In addition to the token properties listed in "[Token Declarations](#)" you must specify these properties for the XML parser:

- **token[x].expression**—Specifies the XPath/XQuery path expression that is traversed to obtain the value for the token. This is a mandatory property.
For example, **token[0].expression=audits/audit/startDate**
- **token[x].node**—Specifies the context node (root node, hop node, or trigger node) relative to which the path expression is evaluated. A context node can be a hop node or a root node. If this property is not specified, it defaults to the trigger node.

For example, **token[0].node=host**

Extra Events

Optional. If you need your FlexConnector to collect different event types for the same trigger node or from different trigger nodes, you can use this property to specify other XQuery configuration files in the current

configuration file.

To specify extra events, use these properties:

- **extraevent.count**—Specifies the number of extra events; for example, **extraevent.count=2**
- **extraevent[x].filename**—Specifies the file name of the additional configuration file that this parser should use; for example, **extraevent[0].filename=ncircle_xml_file/ncircle_xml_file.xml**
- **extraevent[x].name**—Specifies a name to associate with the extra events; for example, **extraevent[0].name=/scanner/device/uri/aggregated**

Examples of Token Mappings

A token captured from the root node: **token[0].expression=audits/audit/startDate**

- A token captured from the hop node 1:

```
token[2].name=ip
token[2].type=IPAddress
token[2].expression=ip
token[2].node=host
```

- A token captured from the hop node 2:

```
token[5].name=protocol
token[5].expression=protocol
token[5].node=vulnref
```

- A token captured from the trigger node, when **token[x].node** is specified:

```
token[8].name=name
token[8].expression=name
token[8].node=
```

- A token captured from the trigger node, when **token[x].node** is not specified:

```
token[13].name=descr
token[13].expression=description
```

Prepare to Write the Parser - Identify Namespace, Nodes, and Tokens

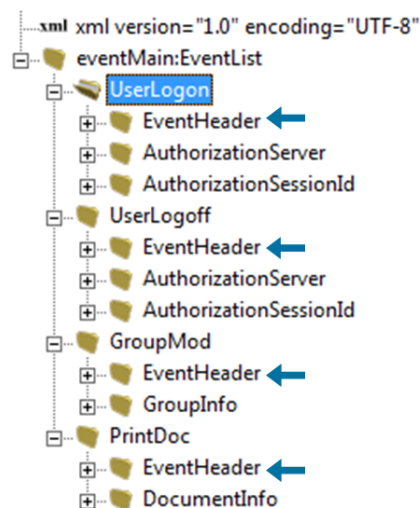
Before writing the parser, examine your source XML log files and complete the following tasks:

- [Find the Trigger Node - the Most Important Step](#)
- [Decide if You Need a Namespace](#)

- [Identify Hop Nodes](#)
- [Identify Tokens](#) (including attributes and nodes as needed)

Find the Trigger Node - the Most Important Step

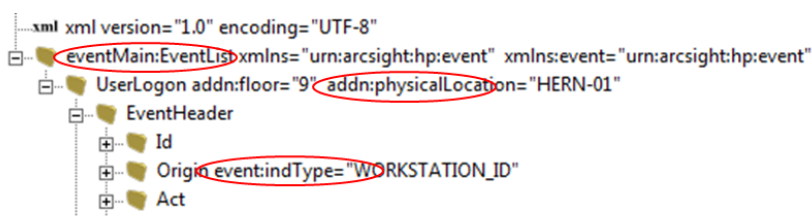
Look at the XML log file and find which node that all events have in common. When you determine this, you can use this node as the trigger node. The trigger node will generate events. In the XML example below, the trigger node identified is **EventHeader**:



Decide if You Need a Namespace

You will need a namespace if a namespace is declared in the header of your XML source file. If you find an element or a node with a colon (:) in its name, the first part of that element or node is its namespace, and must be declared in the parser. See "[Namespace](#)" for more information.

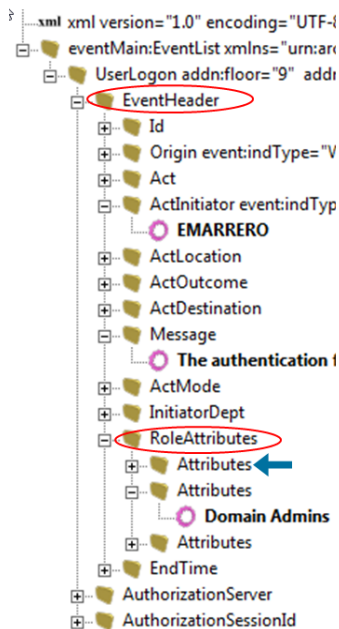
In the following example, elements with colons are circled.



Identify Hop Nodes

Optionally, identify which node or nodes other than the trigger node contain relevant information for security events. See "[Hop Nodes](#)" for details.

In the following example, the trigger node, Attributes, is indicated by an arrow.



In this example, the trigger node is Attributes, so the hop nodes could be:

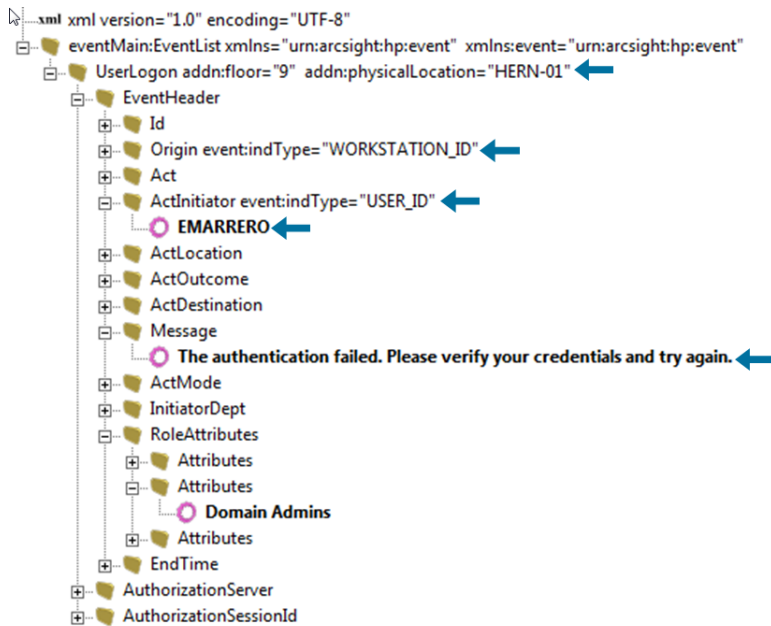
```
hop.node.count=2
hop.node[0].name=header
hop.node[0].expression=//EventHeader
hop.node[1].name=role
hop.node[1].expression=$header//RoleAttributes
```

You can think of hop nodes as variable declarations for long expression paths. For example, if you have to jump three nodes down before finding the trigger and the information to be parsed, you can declare this a named "constant" path in the hop nodes. You can then use this as a variable for the token expression instead of typing the entire path repeatedly.

Identify Tokens

Identify which information you want to extract from each event. Tokens are attributes or text under any node. See "[Token Declarations](#)" for more information.

In the following example, tokens are identified by arrows.



Create the XML FlexConnector Parser

To create the parser, use the information on namespaces, hop nodes (not used in this example), trigger nodes, and tokens you gathered when you examined the source XML file.

- [Parser Development - First Several Lines](#)
- [Parser Development Continued - Tokens](#)
- [Parser Development Continued - Mappings](#)
- [Categorization](#)
- [Copy the Parser Into the Folder](#)

Parser Development - First Several Lines

This is an example of the top portion of a parser:

```
namespace.count=4
namespace[0].prefix=default
namespace[0].uri=urn:arcsight:MF:event
namespace[1].prefix=event
namespace[1].uri=urn:arcsight:MF:event
namespace[2].prefix=addn
namespace[2].uri=urn:arcsight:MF:addn
namespace[3].prefix=eventMain
namespace[3].uri=urn:arcsight:MF:event:main
```

Tokenization Section

```
trigger.node.expression=//EventHeader
additionaldata.enabled=true
token.count=25
```

"[Parser Development Continued - Tokens](#)" contains examples of tokens, and continues after the line `token.count=25`.

Parser Development Continued - Tokens

Use the XML tools listed in "[XML Tools](#)" to create expressions for the tokens. All expressions are relative to the trigger node. Expressions are shown in the example below, which is a continuation of the parser started in the previous section:

```
## Tokenization of Event Type (Root Element) Section
token[0].name=eventType
token[0].expression=../name()
token[1].name=eventPhysicalLocation
token[1].expression=../@addn:physicalLocation/string()
token[2].name=eventFloorLocation
token[2].expression=../@addn:floor/string()

## Tokenization of EventHeader Section
token[3].name=eventNetwork
token[3].expression=@event:network/string()
token[4].name=eventId
token[4].expression=Id
token[5].name=Origin
token[5].expression=Origin
token[6].name=OrgType
token[6].expression=Origin/@event:indType/string()
```

Parser Development Continued - Mappings

Map tokens to event fields, and add severity mappings. Note that all unmapped tokens are passed as additional data fields. See the following example:

```

# Regular Mapping Section
event.deviceEventClassId=eventType
event.name=Action
event.deviceSeverity=Outcome
event.message=Mensaje
event.sourceNtDomain=Dept
event.deviceFacility=Mode
event.deviceExternalId=SessionId
event.endTime=DateTime
event.deviceCustomString1Label=__stringConstant("Physical Location")
event.deviceCustomString1=__concatenate(DestLocation,"|",DestFloor)
event.deviceCustomString2Label=__stringConstant("Attributes")
event.deviceCustomString2=Attributes
event.deviceCustomString3Label=__stringConstant("Auth Server")
event.deviceCustomString3=AA Server
event.fileName=Doc

# Connector Constants
event.deviceVendor=__getVendor("BANANA")
event.deviceProduct=__stringConstant("XMLSCHEMA")
event.deviceVersion=__stringConstant("1.0")

# Severity Mapping Section
severity.map.high.if.deviceSeverity=DENY
severity.map.medium.if.deviceSeverity=FAILURE
severity.map.low.if.deviceSeverity=SUCCESS, GRANT

```

Note that severity mappings are often overlooked, and are key to event normalization. These mappings are required. See ["Severity Mapping"](#) for details on adding severity mappings.

Categorization

Add categorization to your parser. This section is required. This is an area that is often overlooked, and is important because categorization is used for event normalization. For example:

```

event.deviceEventClassId,event.deviceSeverity,set.event.categoryObject,set.event.categoryBehavior,set.event.
categoryTechnique,set.event.categoryDeviceGroup,set.event.categorySignificance,set.event.categoryOutcome
UserLogoff,SUCCESS,/Host/Application,/Access/Stop,/Policy,/Application,/Normal,/Success
GroupMod,SUCCESS,/Host/Application,/Modify/Attribute,/Policy,/Application,/Suspicious,/Success
UserLogon,SUCCESS,/Host/Application,/Access/Start,/Policy,/Application,/Normal,/Success
PrintDoc,SUCCESS,/Host/Application,/Execute,/Policy,/Application,/Informational/Alert,/Success
UserLogoff,FAILURE,/Host/Application,/Access/Stop,/Policy,/Application,/Informational/warning,/Failure
GroupMod,FAILURE,/Host/Application,/Modify/Attribute,/Policy,/Application,/Informational/warning,/Failure
UserLogon,FAILURE,/Host/Application,/Access/Start,/Policy,/Application,/Informational/warning,/Failure
PrintDoc,FAILURE,/Host/Application,/Execute,/Policy,/Application,/Informational/warning,/Failure

```

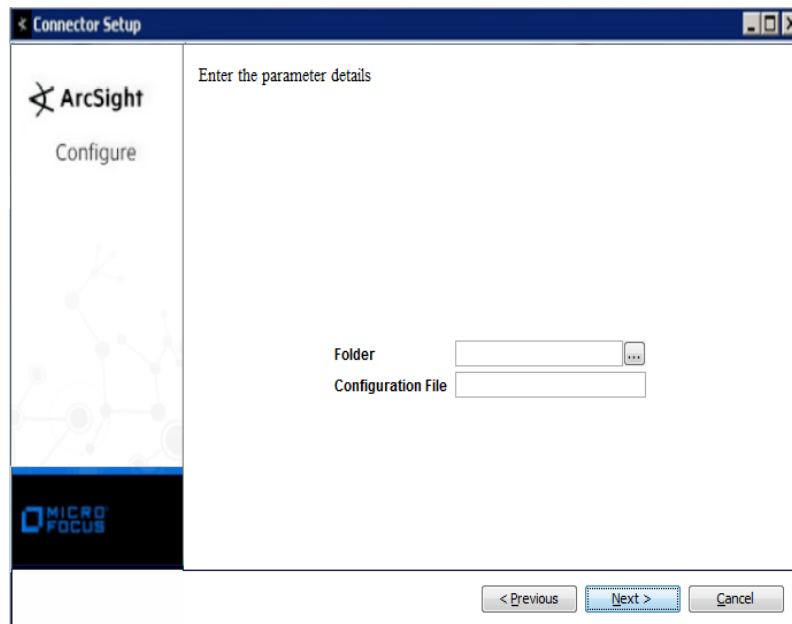
For more about the possible values, see the "Categories" topic in the Console Help or the *ArcSight Console User's Guide*. Also, see ["FlexConnectors and Categorization"](#).

Copy the Parser Into the Folder

After you develop the parser file, you must copy it into this location: **ARCSIGHT_HOME\current\user\agent\flexagent**. This is the required location of the custom parsers you develop for the FlexConnector.

Install the FlexConnector

To install a connector to parse event information presented in standard XML schema, select **ArcSight FlexConnector XML File** from the list of SmartConnectors to install.



Parameter	Description
Folder	The absolute path of the directory where log files for the FlexConnector are located. For example: c:\logs
Configuration File	The base name of the configuration file that describes the format of the log file. The suffix .xqueryparser.properties is appended automatically. For example, if you specify: log The filename becomes: ARCSIGHT_HOME\user\agent\flexagent\log.xqueryparser.properties

Run the connector either as a service or standalone.

Appendix K: Frequently Asked Questions

For general troubleshooting information, participate in the ArcSight user community located at <https://community.softwaregrp.com>. Many questions are answered there.

Where should I look for FlexConnector error and warning messages?

Examine the `agent.log` file located in `ARCSIGHT_HOME\current\logs`. Look for lines containing `[ERROR]` and `[WARN]`.

What data types are supported in SQL server database?

These are the supported SQL server data types. For other data types, the **CASTing** function might be required.

`tinyint`
`bit`
`smallint`
`numeric`
`bigint`
`varbinary`
`float`
`char`
`varchar`
`nvarchar`
`ntext`
`timestamp`

Why does my connection to SQL Server fail/hang?

Check `agent.log` for the underlying error as the user interface error does not show the root cause. Oracle has released Java 6 update 30 (6u30) that behaves differently from JRE 6u29, causing possible database connection problems for SQL Server database connectors using JDBC connection. These connection problems can occur with JRE 1.6.0_29 (6u29) and later versions.

Microsoft recommends using JRE 6u30 (and above) instead of JRE 6u29. Apply the "SQL Server 2008 R2 Service Pack 1 Cumulative Update 6" patch to the SQL server if you are experiencing connection failures or hangs.

How do I define multiple trigger nodes for an XML FlexConnector?

Specify multiple triggers. To do so, specify each trigger node in its own properties file, with one for each extra event or trigger node.

Is it possible to have SNMP traps with the same OID but different meanings (for example, for login failed or user created)? In such cases, can I use the SNMP connector?

Yes. If there are different trap types, define one parser for each trap type and define the field assignments differently within each parser.

I have successfully developed a FlexConnector with a connector type daemon but now need to change the connector type from Syslog daemon to Syslog file. How do I implement this change?

Use the same properties files in the same location, but remove the `agent.properties` file from `user/agent`. Re-install it as a Syslog File Connector.

Can host names take values with spaces?

No. Host names that include spaces are invalid.

Is there a way to perform a one-time query to get past events?

Yes, set the `startatdate` parameter in `ARCSIGHT_HOME\current\user\agent\agent.properties` file, as follows:

For a time-based FlexConnector:

```
agents[x].startatdate=01/01/1970 00:00:00
```

For an ID-based FlexConnector:

```
agents[x].startatid=0
```

My database has date and time in two columns. How can I map this to a timestamp?

The two columns will need to be concatenated and possibly converted to strings using SQL functions so that they can be mapped to a single ArcSight event field.

What does the error “Unable to detect DB version” mean?

This error indicates that the connector property `version.query` in the configuration file is invalid, returns no data, or there is a database connection problem.

Are there best practices for writing regular expressions?

Try to be as specific as possible. For example, to parse a string `"abc,def,ghi,"` do not use:

```
".*,.*,.*" or
```

```
".*?,.*?,.*?" or
```

```
"\\S+,\\S+,\\S+"
```

Instead, use:

```
"[^,]+,[^,]+,[^,]+"
```

This is because the first examples will cause the pattern matcher to compute all the possibilities. Of course, if the string is space-separated, `S+` makes sense.

The `.*` expression is not recommended. Never use more than one of these in a regular expression, preferably at the end. A question mark (`?`) is also not recommended. Never use more than one.

How do I parse a timestamp in the RFC 5424 format?

Use `"T"` in the timestamp, which represents the RFC 5424 syslog time format. For example:

```
2012-01-17:2012-01-17T10:39:32+08:00
```

with this format

```
yyyy-MM-dd'T'HH:mm:ssZ
```

Should I include comments in my connector configuration (parser) file?

Yes, comments can be helpful. Use the `#` sign at the beginning of each comment line to indicate that it is a comment. You can also include some sample events in your comments that you used to help you write the parser.

How do I keep track of the contents of device custom string field?

If you are populating `deviceCustomString1`, fill in `deviceCustomString1Label=stringConstant` (describe the contents of `deviceCustomString1`). If a number of bytes go into `deviceCustomString1`, then `Number of Bytes` must be included in `deviceCustomString1Label`.

How can I identify my events?

Add `deviceVendor`, `deviceProduct`, `deviceVersion` to your configuration file.

How do I specify format for a datestamp extracted from a file name?

In `agent.properties`, add the format after the field your are populating. For example:

```
agents[0].extractfieldnames=deviceCustomDate1(yyyymmdd)
```

Can FlexConnectors directly read compressed files (such as .zip)?

Yes.

What can I do if events are not being collected?

If an event or events are not being collected, include `do.unparsed.events=true` in the configuration file.

Where can I find errors and messages related to FlexConnector operation?

Examine the `agent.log` file to look for errors and warnings.

I cannot always find Regex Tool errors. Where do some of the Regex Tool errors appear?

FlexConnector Regex Tool can write errors to the command window where the tool was launched.

How can I enable debug mode logging for a FlexConnector?

Enabling debug mode logging increases the amount of FlexConnector log information. With debug mode logging enabled, the **agent.log** files are created quickly, so limit the amount of time the FlexConnector is in debug mode to 10 to 15 minutes.

To enable debug mode logging:

1. Add the following two lines to **ARCSIGHT_HOME\current\user\agent\agent.properties**:
 - **log.global.debug=true**
 - **log.channel.file.property.package.com.arcsight=0**
2. After you complete your troubleshooting, remove the two above lines from the **agent.properties** file.

Send Documentation Feedback

If you have comments about this document, you can [contact the documentation team](#) by email. If an email client is configured on this computer, click the link above and an email window opens with the following information in the subject line:

Feedback on Developer's Guide (Connectors 7.13.0)

Just add your feedback to the email and click send.

If no email client is available, copy the information above to a new message in a web mail client, and send your feedback to arcsight_doc@microfocus.com.

We appreciate your feedback!