Collector Guide

HawkEye AP 6.1.0

August 28, 2015

Hexis Cyber Solutions, Inc.
2800 Campus Drive, Suite 150
San Mateo, CA 94403
+1 855 529-3929
www.hexiscyber.com

TABLE OF CONTENTS

PREFACE

The *Event Collection Guide* describes how batch event data is collected and stored in version 6.0.1 of a HawkEye AP deployment. Log entries are collected and stored through the HawkEye AP batch collection, which processes file-based event logs and loads their entries into the EDW.

This preface contains these sections:

AUDIENCE FOR THIS BOOK

The audience for the *Collector Guide* includes system administrators who initially set up receivers, retrievers, parsers, and the Collector. The audience also includes developers who create Parser rules and Parse, Transform and Load (PTL) files.

EVENT COLLECTION GUIDE ORGANIZATION

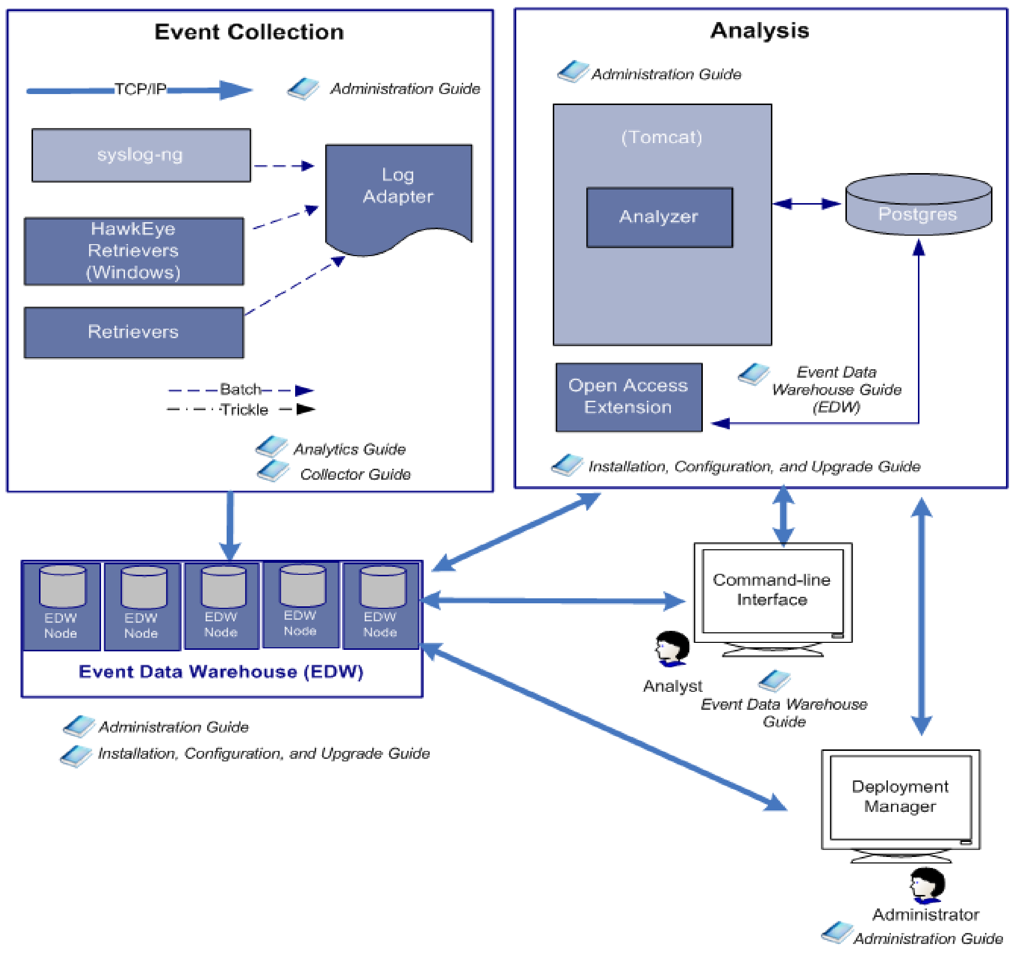
The information in the *Collector Guide* is organized into the following chapters:

- Chapter 1: Introduction—Describes the batched event-log data and illustrates the general architectures of receivers and the Collector.
- Chapter 2: Collector Configuration—Describes how to initially set up the Collector and its different kinds of modules: retrievers, preprocessors, and loaders; describes how to establish several Collectors in a *daisy-chain* configuration to load event-log data into an Event Data Warehouse (EDW).
- Chapter 3: Configuring Retrievers and Receivers—Describes how to set up and configure HawkEye AP receivers for different streaming protocols: syslog, SNMP, and LEA.
- Chapter 4: HawkEye Retriever Configuration—Describes the standard and advanced configuration properties for the Hawkeye Retriever, as well as the procedure for configuring agentless authentication.
- Chapter 5: Configuring Netflow Receiver for Data Collection and Analysis—Describes how to configure Netflow Receiver, the HawkEye AP process of collecting and analyzing network.
- Chapter 6: Configuring Common Event Format (CEF) Forwarder—Describes the how to configure the CEF Forwarder, the HawkEye AP process for converting event data to the CEF format and forwarding this data to the ArcSight System for real-time processing.
- Appendix A: PTL File Format—Describes how to write a Parse, Transform and Load (PTL) file used to transform event data into normalized data for storage in the Event Data Warehouse (EDW).

ROAD MAP TO HAWKEYE AP DOCUMENTATION

This document, the *Collector Guide*, is part of the larger documentation set of your HawkEye AP system. Figure P-1 illustrates HawkEye AP components and modules in the context of their function within the HawkEye AP system.

Figure P-1: Road Map to HawkEye AP Documentation



The table below describes all the manuals in the HawkEye AP documentation set and the user roles to which they are directed.

Role	Tasks	Documentation
Business Analyst or System Administrator	• Learn about Analytics • Configure Insider Threat (IT) Detection Model • Use IntelliSchema views • Learn about the Foundation and Compliance Analytics packages • Learn about additonal Analytics packages	*Analytics Guide*

Role	Tasks	Documentation
Developer, Report Developer or Security Analyst	• Use HawkEye AP SQL, HawkEye AP SQL functions, and libraries to create reports or query the EDW • Access EDW data using open standards as ANSI SQL, ODBC, and JDBC • Create and use Perl code in HawkEye AP SQL statements • Use the DBD Driver to query HawkEye AP from other locations	*Event Data Warehouse Guide*
Security System Administrator	• Configure retrievers, receivers, and collectors • Enable/disable log adapters • Configure HawkEye Retriever • Create log adapter PTL files	*Collector Guide*
System Administrator	• Install HawkEye AP • Configure HawkEye AP and its components • Configure VMWare	*Installation, Configuration, and Upgrade Guide*
System Administrator	• Manage the HawkEye Event Data Warehouse (EDW) • Manage the Collector • Manage users, groups, and permissions • Archive to nearline storage • Monitor security alerts • Monitor logs • Troubleshoot • Error Messages	*Administration Guide*
Legal	Monitor third-party licenses	*Third-Party Open Source Licensing*

TIP: You can access the manuals listed above from:

- HawkEye AP Welcome page

  Click the **Documentation** hyperlink.

CONVENTIONS USED IN HAWKEYE AP DOCUMENTATION

This convention...	Indicates...	Example
bold text	Names of user interface items, such as field names, buttons, menu choices, and keystrokes	Click **Clear Filter**.
italic text	Indicates a variable name or a new term the first time it appears	`http://<host>:<port>/index.mhtml`
`Courier text`	Indicates a literal value, such as a command name, file name, information typed by the user, or information displayed by the system	`atquery localhost:8072 myquery.sql`

This convention...	Indicates...	Example
SMALL CAPS	Indicates a key on the computer keyboard	Press ENTER.
`{ }`	In a syntax line, curly braces surround a set of options from which you must choose one and only one. **NOTE**: Syntax specifications for SELECT statements include curly braces as part of the `{INCLUDE_BAD_LOADS]` keyword.	`{ start | stop | restart }`
`[ ]`	In a syntax line, square brackets surround an optional parameter	`atquery [options] <`*`host`*`>:<`*`port`*`> -`

`|`	In a syntax line, a pipe within square brackets or curly braces separates a choice between mutually exclusive parameters
	NOTE: Syntax for defining a Nearline Storage Address (NSA) includes a pipe.
		`{ start | stop | restart }`
		
		`[g|m]`

`...`	In a syntax line, ellipses indicate a repetition of the previous parameter	The following example indicates you can enter multiple, comma-separated options: `<`*`option`*`>[, <`*`option`*`>[…]]`

backslash (\)	A backslash in command-line syntax or in a command example behaves as the escape character on Unix. It removes any special meaning from the character immediately following it. In HawkEye AP documentation, a backslash nullifies the special meaning of the newline character as a command terminator. Without the backslash, pressing ENTER at the end of the line causes the Unix system to execute the text preceding the ENTER. Without the backslash, you must allow long commands to wrap over multiple lines as a single line.	`atquery --user=administrator \`
		`--pass=pass:p@ss localhost:8072\`
		`-e='SELECT * FROM system.users;'`

CONTACTING TECHNICAL SUPPORT

For additional help, email support@hexiscyber.com or call +1 855 529-3929. Also see the Hexis Cyber Solutions Technical Support web page at http://www.hexiscyber.com/content/ for HawkEye AP documentation, product downloads, and additional information on contacting support to escalate help on issues that impact your production environment.

CHAPTER 1

Introduction

This chapter contains these sections:

- “Overview of Collecting Event Data”, next
- “Processing Batched Events”, next
- “Collecting and Analyzing Network Data with the Netflow Receiver”, on page 18

OVERVIEW OF COLLECTING EVENT DATA

Before your enterprise can analyze event log data, original event-log data must be collected into your HawkEye AP system.

Your HawkEye AP system supports batch data collection in which events are collected from log files and other event repositories maintained by network devices and software applications. For more information on how to collect batches of log entries, see “Processing Batched Events”, next.

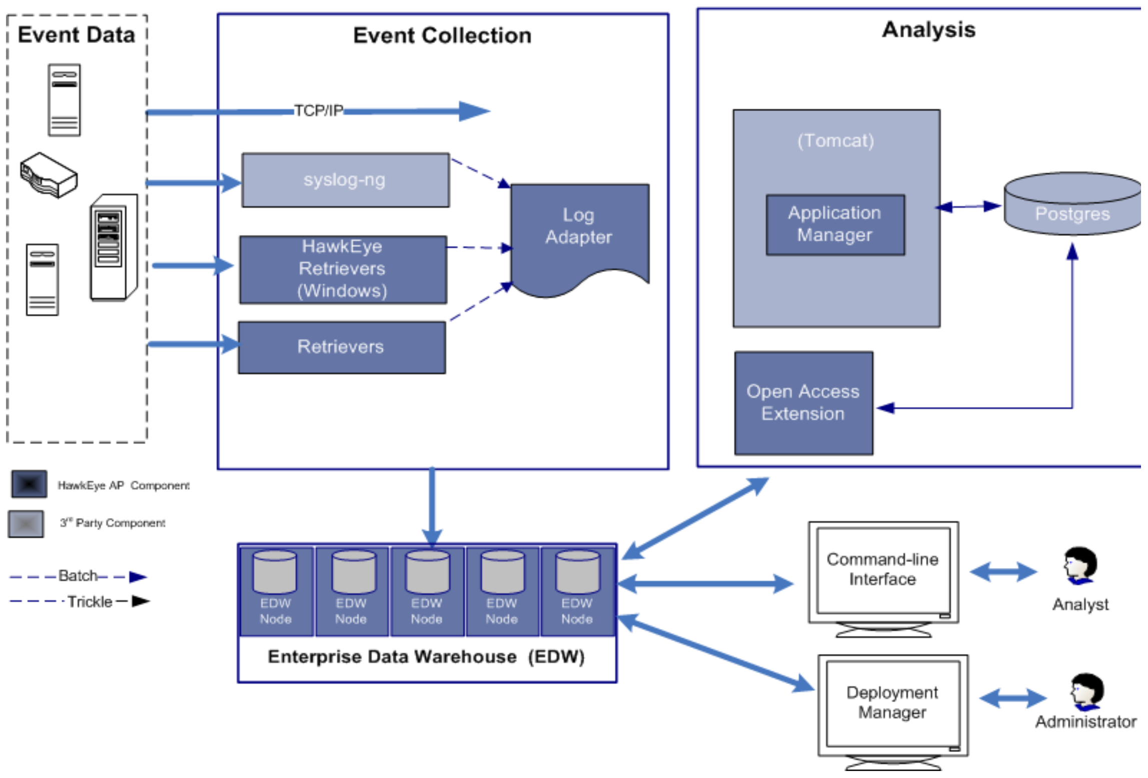
Regardless of how you collect the data, collection sometimes fails. Failures can occur anywhere between the source of the log records and the Event Data Warehouse (EDW). For example, the log source system might be down, or one of the HawkEye AP components or processes (such as the Collector or a receiver) might not be running, or there might be a communication failure that prevents loading of the log data. Alternatively, there may have been an upgrade of the software on the log-source system that prevents parsing of the log records.

If undetected, collection failures leave gaps in the historical event record stored in the EDW. A source failure occurs when the volume of data loaded into the EDW from a particular log source varies from the expected amount. The HawkEye AP Analyzer provides ways to track the health of specified log sources. For more information, see the Analyzer documentation.

PROCESSING BATCHED EVENTS

As shown in Figure 1-1: the Collector polls data sources or repositories to retrieve event data, which it loads into the EDW The EDW makes the event data available for reporting. Administration users can access EDW data by using the command-line interface (CLI) utilities on a Linux system.

Figure 1-1: HawkEye AP Architecture for Event Processing



About the Collector

The Collector is a component of HawkEye AP software that collects events in batches from log files. It automates the collection and loading of event-log data into the Event Data Warehouse (EDW), either scheduled, on-demand, or in continuous mode.

This section describes these topics:

- "Collector Highlights", next
- "Collector Architecture", on page 15
- "Collector Part Descriptions", on page 15

Collector Highlights

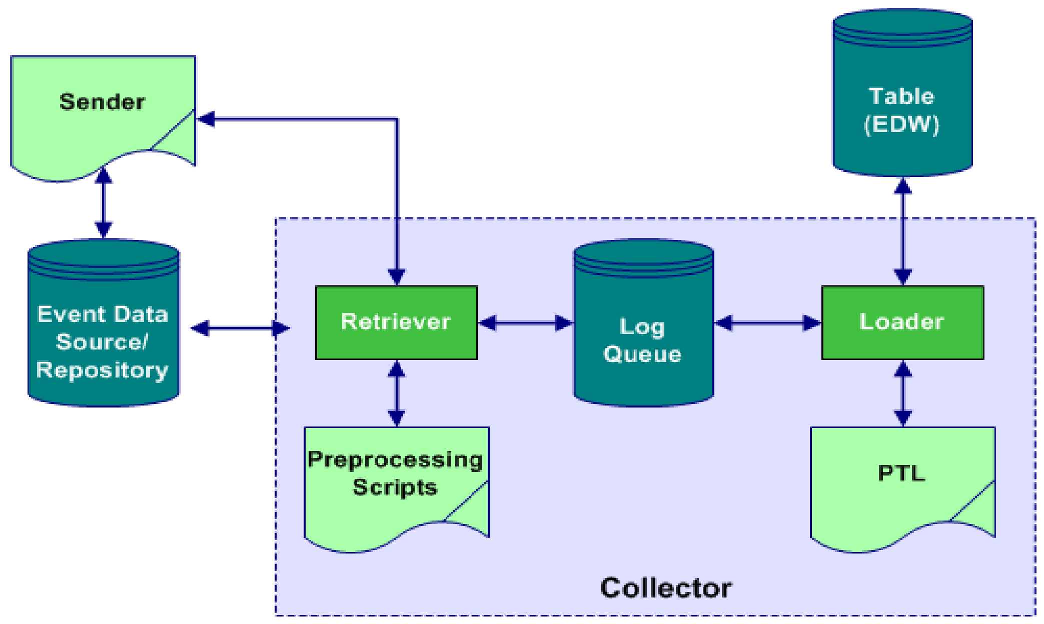
These are the key features of the Collector:

- Retrieves logs from one or multiple sources
- Triggers backups and preprocessing on source log files
- Loads log entries into EDW tables
- Supports standard interfaces, including TCP, FTP, SFTP, SCP, LEA, SMB, RCP, HTTP (Get and Push)
- Provides status of retrieval and loading activities through the Collector activity logs

- If configured, sends email alarms that notify administrators of problems with the system
- Command-line interface
- Enables loading of log entries according to a specified schedule
- Runs with minimal administration

Collector Architecture

The graphic below illustrates the modules that participate in the retrieval and loading of event-log data into the EDW.



The Collector is configured to start up one or more Retrievers. The Retriever gathers the log data from the source, passes it through preprocessors, and places the resulting log file in one of the Log Queues. For remote systems using HTTP Push, the *collector_push* script must be started to assist in the retrieval of the log data.

The Collector is also configured to start up one or more Loaders. A Loader takes a log file from the Log Queue, parses and transforms the log file as specified by the PTL (Parser, Transform, and Load), and loads the log data into a specified EDW instance, namespace, and table.

While the modules work together to move the log data from the source to an EDW table, each module runs in isolation. As a result, each module can be configured and scheduled independently.

Collector Part Descriptions

These are the different parts that the Collector manages in order to collect and load log entries from batched log files.

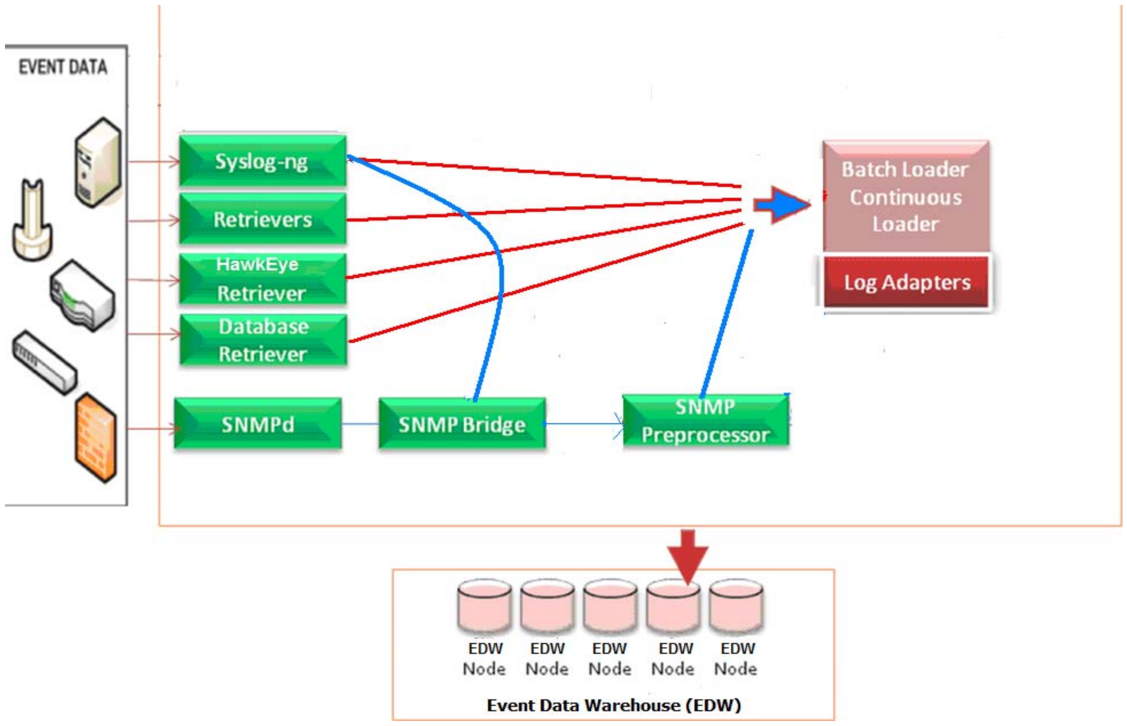
Module	Description
Collector	The overall system for the gathering and loading of event-log data into an EDW instance. Its core process is responsible for reading the configuration file, maintaining state, starting Retrievers and Loaders, and managing Log Queues.
Loader	A process spawned by the Collector that loads data from a log file into an EDW table after parsing and transforming the log data as specified by a PTL file.
PTL File	A file that provides the EDW with instructions for parsing a specific log file format and for targeting the parsed fields to a specific table schema. PTL files with international character data must be written with UTF-8 encoding.
Log Queue	A user-specified directory where log files are stored while being downloaded, while waiting to be loaded, or during loading. The log-file naming convention used ensures consistency of state and prevents concurrency or locking issues. By default, log files in the queue are expected to be written with UTF-8 encoding; other character encoding schemes can be specified.
Log File	A file in the Collector system containing log data.
Retriever	A process spawned by the Collector that retrieves log data (either directly or by using a Sender), performs any required preprocessing, and writes the log data to log files in specified Log Queues.
Sender	A process sitting on a remote system that sends (or pushes) the log data to a location where it can be received by a Retriever.

SNMP Bridge for Transport

For batch parsing and loading, HawkEye AP uses the SNMP Bridge to trap and send messages into the syslog-ng server. The SNMP Bridge is created through the configuration of the standard Linux SNMP Daemon (**snmpd**) to log SNMP Traps to a Linux named pipe; a second program, **snmptrap-bridge**, reads from the named pipe, ensuring that each SNMP Trap is encoded as a single line of text. After the pipe is read, the **snmptrap-bridge** program then forwards the text to the local SYSLOG (using the same "packetizing" logic that the EDW uses if the message is too large for a single SYSLOG frame to carry).

Thus, syslog-ng forwards SNMP Traps into SYSLOG. Once a message is in SYSLOG, syslog-ng retrieves the message and treats it as any other SYSLOG message. This SNMP Bridge configuration is shown in Figure 1-2.

Figure 1-2: SNMP Bridge Configuration



Using SNMP Traps with SYSLOG

There is no configuration required for the SNMP Bridge when using SYSLOG as a destination. To use SNMP Traps, choose one of the following alternatives:

Perform the following steps to trap messages for parsing and loading:

1. Configure syslogn-ng to direct the SNMP Trap data (via SYSLOG) into a log file.

2. Configure the collector to load that log file into some table in the EDW.

3. Create a PTL script to parse the log file.

Enabling Another type of Trap in the SNMP Bridge

To enable another type of trap in the SNMP Bridge, configure the **$BASE/etc/sysconfig/snmptrap-bridge.options** file by specifying trap type as highlighted in the TCP entry below:

```
#if [ -e $BASE/etc/sysconfig/snmptrap-bridged.options ]; then
if [ -e $BASE/etc/sysconfig/snmptrap-bridge.options ]; then
. $BASE/etc/sysconfig/snmptrap-bridge.options
else
#OPTIONS="-f $BASE/var/run/$PROG.fifo -l $BASE/var/lock/subsys/$PROG -p $BASE/
var/run/$PROG.pid -s user.info"
#OPTIONS="-f $BASE/var/run/$PROG.fifo -l $BASE/var/lock/subsys/$PROG -p $BASE/
```

```
var/run/$PROG.pid -s user -d syslog:info"
#OPTIONS="-f $BASE/var/run/$PROG.fifo -l $BASE/var/lock/subsys/$PROG -p $BASE/
var/run/$PROG.pid"
OPTIONS="-f $BASE/var/run/$PROG.fifo -l $BASE/var/lock/subsys/$PROG -p $BASE/
var/run/$PROG.pid -s user -d tcp:127.0.0.1:5144"
```

About Log Adapter and PTL Files

The HawkEye AP System processes event logs from a variety of hardware and software applications. These sources typically log many types of events that are stored in a variety of formats. A log adapter transforms these diverse events and formats into useful data that can be stored in the HawkEye EDW. Data stored in the EDW is available for analysis using HawkEye AP Analyzer. For more information, see the HawkEye AP Analyzer Documentation.

Log adapters use *PTL files* to define the transformation of log data into data stored in the EDW. Hexis Cyber Solutions provides log adapters with your HawkEye AP distribution for a variety of common information systems and devices. For more information, see the Log Adapters References for each Log adapter, available in the HawkEye AP documentation set. Additional log adapters are also available from Sensage Technical Support. You can also create a log adapter using the PTL file format. For more information, see Appendix A: PTL File Format.

COLLECTING AND ANALYZING NETWORK DATA WITH THE NETFLOW RECEIVER

The Netflow Receiver provides Network Managers a detailed view of application flow on the Network. Routers and Layer 3 switches act as sensors pushing data to Netflow in a form of telemetry. The routers provide the information on network traffic flow, while micro-analytical tools that include an IDS (Intrusion Detection System) examine packet contents. Thus Netflow provides monitoring on who talks to who, for how long, at what intervals, on which ports and protocols, and how much data is being exchanged.

By capturing Netflow data, network traffic becomes visible, providing a defense against DoS attacks and other forms of undesireable traffic. Wtih the information on hand, you can provide the appropriate response to network-based threats, and gather valuable intelligence about application usage for capacity planning.

For details on Netflow Receiver, including configuration, see “Configuring Netflow Receiver for Data Collection and Analysis”, on page 123

CHAPTER 2

Collector Configuration

This chapter contains the following sections:

ROOT DIRECTORIES USED BY THE COLLECTOR

This section describes special directories that the Collector uses for special purposes:

The following topics show examples of the XML elements in the Collector configuration file that specify these special directories. The Collector configuration file is found in this location:

```
<HawkEye AP Home>/etc/collector/config.xml
```

NOTE: If you change the value for a special directory in `config.xml` and the directory does not exist, the Collector creates it the next time the Collector starts.

FileRoot

```
<FileRoot><collector-datadir></FileRoot>
```

Root directory for all volatile files used by the Collector, including temporary log files and state files. You configured the FileRoot directory when you ran the UI Install Wizard and specified a value for the collector-datadir flag.

IMPORTANT: FileRoot must not exist on an NFS-mounted file system.

StateRoot

```
<StateRoot>state</StateRoot>
```

Location for state files. The path is relative to FileRoot if no leading slash (/) is present; otherwise, the path is absolute, beginning from the root of the local file system. In general, you do not change this setting; the install and configuration process sets it automatically.

CLIRoot

```
<CLIRoot><HawkEye AP Home>/bin/</CLIRoot>
```

Location for EDW command-line tools, such as `atload`, which loads event-log data into EDW tables. The path is not relative to FileRoot. In general, you do not change this setting; the HawkEye AP installation process sets it automatically.

The installation process places all command line tools in:

```
<HawkEye AP Home>/bin
```

Configuring Collector Modules

After installing the Collector, you configured its basic properties using the Deployment Manager described in the Installation, Configuration, and Upgrade Guide. In addition to basic properties, such as root directories, you must configure the following Collector modules:

Module	Description	Documentation
Log Queues	Directories on the Collector host where Collector modules store and process log files before loading the event-log data into the EDW	"Configuring Log Queues", on page 25
Retrievers	Collector modules that retrieve source log files and put them into log queues for loading into the EDW	"Defining Retrievers", on page 27
Preprocessors	Scripts or binaries run on source log files captured by Retrievers before putting them into log queues	"Configuring Preprocessors", on page 41
Loaders	Collector modules that load log files from log queues into the EDW	"Defining Loaders", on page 42
Daisy chains	Special Collector configurations that allow the output of one Collector instance to serve as the input to another	"Creating Daisy Chains", on page 51

Configuring Collector Modules

You configure Collector modules by editing the Collector configuration file.

To edit the Collector Configuration file:

1. Log in as the specified Collector user, **hexis** by default.

2. Reload the Collector by running the following script:

```
/opt/hexis/hawkeye-ap/etc/init.d/sensage_collector reload
```

3. If you configured several Collectors in a daisy chain and *did not* specify all Collector hosts in the `collector-hosts` property, run the following command to restart the Collector:

```
<HawkEye AP Home>/etc/init.d/sensage_collector restart
```

For more information, see “Creating Daisy Chains”, on page 51.

4. To configure error logging and email alerts, see “Configuring Error Reporting”, on page 54.

NOTE: The `config.xml` file contains elements for specifying the FileRoot, PTLRoot, StateRoot and CLIRoot directories. Generally they are set automatically during the installation and configuration processes. For more information, see "Root Directories Used by the Collector", on page 27.

Using the CollectorAdmin Tool to Manage your Configuration

The CollectorAdminTool allows you to enable and/or disable log adapters, retrievers , and loaders, and list the current configuration settings in a specific **config.xml** file.

Viewing and Modifying your Configuration

When you use the CollectorAdmin tool, you can:

- List all log adapters, retrievers, and loaders with their status (enabled / disabled)
- List all enabled or disabled log adapters, retrievers, and loaders
- Enable or disable log adapters, retrievers, and loaders
- Show or truncate Collector statistics

CollectorAdmin tool Usage

To use the CollectorAdmin tool, run the following command:

```
<prefix>/bin/collectorAdmin <options>
```

The table below lists the currently available options. Note that you can submit multiple options on the command line.

Command Line Option	Description
all	Show all adapters and enable/disable status.
enabled	Show enabled adapters.
disabled	Show disabled adapters.
find	Locate definition of a loader, retriever or queue
detail	Show all loaders, retrievers, and queues.
vdetails	Show all loaders, retrievers, and queues with paths.
enable [adapter]	Enable the named adapter
disable [adapter]	Disable the named adapter.

Command Line Option	Description
showstatus	Show Collector statistics chart
clearstatus	Truncate the Collector statistics file.

NOTE: Optionally, you can use a manual procedure to enable and disable **config.xml** files for particular adapters. For details, see “Enabling and Disabling Log Adapters Manually”, on page 24.

Examples

1. The following option lists all adapters as well as displays their enabled/disabled state:

```
<prefix>/bin/collectorAdmin all

Loading config files from '/opt/hexis/hawkeye-ap/etc/collector'
Adapter Status-
apache_access_syslogng: disabled
apache_error_syslogng: disabled
bluecoat_sgproxy_batch: enabled
catbird_vsecurity_cef_syslogng: enabled
checkpoint_opsec_lea: disabled
cisco_acs_syslogng: disabled
cisco_asa_syslogng: disabled
```

2. The following option displays a list of all Log Adapters that enabled:

```
<prefix>/bin/collectorAdmin enabled

Loading config files from '/opt/hexis/hawkeye-ap/etc/collector'

Adapter Status-
bluecoat_sgproxy_batch: enabled
catbird_vsecurity_cef_syslogng: enabled
cisco_ios_syslogng: enabled
cisco_netflow_receiver: enabled
cisco_pix_syslogng: enabled
```

3. The following option locates the definition of cisco_pix_syslogng:

```
<prefix>/bin/collectorAdmin find cisco_pix_syslogng

Loading config files from '/opt/hexis/hawkeye-ap/etc/collector'
Items matching: cisco_pix_syslogng

Loader l_cisco_pix_syslogng: enabled
 /opt/hexis/hawkeye-ap/etc/collector/adapters/cisco_pix_syslogng/config.xml
LogQueue q_cisco_pix_syslogng: enabled
 /opt/hexis/hawkeye-ap/etc/collector/adapters/cisco_pix_syslogng/config.xml
Retriever r_cisco_pix_syslogng: enabled
 /opt/hexis/hawkeye-ap/etc/collector/adapters/cisco_pix_syslogng/config.xml
```

4. The folloiwng option displays collectorAdmin details of all loaders, retrievers, and queues:

```
<prefix>/bin/collectorAdmin detail

Loading config files from '/opt/hexis/hawkeye-ap/etc/collector'
Items:
```

```
Loader l_apache_access_syslogng: item disabled
Loader l_apache_error_syslogng: item disabled
LogQueue q_apache_access_syslogng: enabled
LogQueue q_apache_error_syslogng: enabled
Retriever r_apache_access_syslogng: item disabled
Retriever r_apache_error_syslogng: item disabled
```

5. The folloiwng option displays all loaders, retrievers, and queues and the paths of the config file(s) that contain these definitions:

```
<prefix>/bin/collectorAdmin vdetail

Loading config files from '/opt/hexis/hawkeye-ap/etc/collector'
Items:
Loader l_apache_access_syslogng: item disabled
 /opt/hexis/hawkeye-ap/etc/collector/adapters/apache_access_syslogng/
config.xml
Loader l_apache_error_syslogng: item disabled
 /opt/hexis/hawkeye-ap/etc/collector/adapters/apache_error_syslogng/config.xml
LogQueue q_apache_access_syslogng: enabled
 /opt/hexis/hawkeye-ap/etc/collector/adapters/apache_access_syslogng/
config.xml
LogQueue q_apache_error_syslogng: enabled
 /opt/hexis/hawkeye-ap/etc/collector/adapters/apache_error_syslogng/config.xml
Retriever r_apache_access_syslogng: item disabled
 /opt/hexis/hawkeye-ap/etc/collector/adapters/apache_access_syslogng/
config.xml
Retriever r_apache_error_syslogng: item disabled
 /opt/hexis/hawkeye-ap/etc/collector/adapters/apache_error_syslogng/config.xml
```

6. The following option disables the apache_access_syslogng Log Adapter:

```
<prefix>/bin/collectorAdmin  disable apache_access_syslogng

Loading config files from '/opt/hexis/hawkeye-ap/etc/collector'
disabled adapter apache_access_syslogng
```

7. The following option enables the apache_access_syslogng Log Adapter:

```
<prefix>/bin/collectorAdmin  enable  apache_access_syslogng

Loading config files from '/opt/hexis/hawkeye-ap/etc/collector'
enabled adapter apache_access_syslogng
```

8. When it comes to enabling/disabling, you can perform multiple requests at once; for example, you can enable and disable adapters in the config files. For example:

```
<prefix>/bin/collectorAdmin  disable apache_access_syslogng enable
apache_error_syslogng  detail apache_access_syslogng

Loading config files from '/opt/hexis/hawkeye-ap/etc/collector'
disabled adapter apache_access_syslogng
enabled adapter apache_error_syslogng
Items:
Loader l_apache_access_syslogng: item disabled and adapter disabled
Loader l_apache_error_syslogng: enabled
LogQueue q_apache_access_syslogng: adapter disabled
LogQueue q_apache_error_syslogng: enabled
Retriever r_apache_access_syslogng: item disabled and adapter disabled
Retriever r_apache_error_syslogng: enabled
```

NOTE: After this tasks, you must reload the Collector for changes to take effect.

9. The following option will dump internal collector statistics:

```
<prefix>/bin/collectorAdmin showstats

Loading config files from '/opt/hexis/hawkeye-ap/etc/collector'
showStats stateDir=/opt/hexis/hawkeye-ap/data/collector//state/statistics.dat
Retriever Statistics:
 r_bluecoat_sgproxy_batch  Downloads: 1, Bytes: 9973
 r_catbird_vsecurity_cef_syslogng  Downloads: 1, Bytes: 31885
 r_cisco_ios_syslogng  Downloads: 1, Bytes: 36061

Loader Statistics:
 l_bluecoat_sgproxy_batch  Uploads: 1, Bytes: 9505, Records: 20, MatchFailures:
0
 l_catbird_vsecurity_cef_syslogng  Uploads: 1, Bytes: 31885, Records: 99,
MatchFailures: 0
 l_cisco_ios_syslogng  Uploads: 1, Bytes: 36061, Records: 214, MatchFailures: 0
```

10. Depending on the confirmation specified, the following options will either reset internal statistics to 0 or keep the existing statistics:

```
<prefix>/bin/collectorAdmin clearstats

Loading config files from '/opt/hexis/hawkeye-ap/etc/collector'
Warning all accumulated statistics data will be removed.
Are you sure you want to clear collector statistics?
y

collectorAdmin clearstats

Loading config files from '/opt/hexis/hawkeye-ap/etc/collector'
Warning all accumulated statistics data will be removed.
Are you sure you want to clear collector statistics?
n
Not clearing statistics
```

Enabling and Disabling Log Adapters Manually

As an alternative to using the Collector Admin Tool to enable and disable Log Adapters, you can also perform a manual procedure to accomplish this task.

To enable and disable Log Adapters , you change the "enabled" setting for a given Log Adapter’s configuration file. Note that the Collector reads all **config.xml** files under the <*HawkEye AP Home*>/ collector/adapters directory and uses the configuration from all "enabled" **config.xml** files.

For example, assume "unix_sudo_syslogng" Log Adapter is enabled and you want to disable it. To do this:

1. Access **/opt/hexis/hawkeye-ap/etc/collector/adapters/unix_sudo_syslogng/config.xml.**

2. Go to the Collector version parameter and setting and change **enabled =’1’ to ’0**,’ as in:

```
<Collector version='5.0' enabled='0'> /* pre-5.0 behavior.  Does not merge */
```

NOTE: If the enabled attribute is absent, the default is **1**.

3. Run the following script reloads the Collector so that retrievers and loaders match the current configuration:

```
/opt/hexis/hawkeye-ap/etc/init.d/sensage_collector reload
```

CONFIGURING LOG QUEUES

Log queues are directories on the Collector host where Collector modules store and process event-log files. Configure log queues in the `config.xml` file using the following example syntax:

```
<LogQueues>
  <LogQueue name="local" path="queue/local" minMBFree="32"
      encoding="Shift-JIS”/>
  <LogQueue name="remote" path="queue/remote" minMBFree="1024"/>
</LogQueues>
```

NOTE: You must configure a different log queue for each loader that you define. In other words, you cannot share a log queue among loaders.

For more information on loaders, see “Defining Loaders”, on page 42.

Use the `<LogQueue>` element within the `<LogQueues>` list to define one or more log queues. These are the attributes of the `<LogQueue>` element:

- **name**—(Required.) Unique name for the log queue
- **path**—(Required.) Location on the local file system where the log files in the log queue are stored, relative to FileRoot.

  For more information on FileRoot, see “Root Directories Used by the Collector”, on page 19.
- **minMBFree**—(Required.) minimum free disk space in megabytes. If the minimum free disk space is not available, the Collector suspends the retrieval process until enough free space is available. You should specify a number that is equal to about a day's worth of log data on your system.
- **encoding**—(Optional.) character encoding of log files in the queue. All log files in the queue can use any character set supported by EDW. This includes encoding schemes that are distributed with Linux such as 'EUC-JP', 'Shift-JIS', 'UJIS', and 'UTF-16'. For a current listing, run the `iconv -l`. The default encoding is UTF-8.

Life Cycle of a Log File in a Log Queue

When the Collector processes incoming event data, it writes the data to a log file that is named by conventions established in each collection process (for example, file naming for syslog sources is defined in the `syslog-ng.conf` file). As the log file is processed by the Collector, this original file’s extensions are renamed and the file is moved to various sub-directories.

Figure 2-1 shows the following processing steps:

1. The Collector writes incoming log data into a file named with the `.out` extension, located in the `<LogQueue>/spool` directory.

2. The Retriever renames the file's extension from `.out` to `.pmd` and begins to calculate an `MD5` checksum for the file. The file is located in the `<LogQueue>/spool` directory. The retriever creates a new file with the same name, but having the `.meta` extension and saves it in the `<LogQueue>/spool` directory. This `.meta` file accompanies all versions of the log file that are moved or renamed by the next steps in the log file's lifecycle.

3. After the `MD5` checksum is calculated, the Retriever renames the `.pmd` file into a file with the `.org` extension, located in the `<LogQueue>/spool` directory.

4. If a backup directory is specified for the log queue, the Retriever copies the file to the specified backup directory (see "BackupDir", on page 30).

After creating any specified backup copies, the Retriever renames the file's extension from `.org` to `.raw`. The file is located in the `<LogQueue>/spool` directory.

5. If a preprocessor is defined (see "Preprocess", on page 30), the preprocessor renames the file's extension from `.raw` file to `.in` and begins to process the file. The preprocessor writes its output to a file with the `.out` file extension, located in the `<LogQueue>/spool` directory. This file grows as the output is generated by the preprocessor.

When the preprocessor completes processing the `.in` file, the Retriever moves the `.out` file to the `<LogQueue>/`directory and renames the file's extension from `.out` to `.log`.

If the preprocessor fails, the log file's extension is renamed to `.noproc` and the file is moved to the `<LogQueue>/spool` directory. After debugging and correcting the failure, the HawkEye AP administrator should:

a Rename the file with an .org extension.

b Do not move or rename the .meta file.

c Restart the Collector using the Deployment Manager.

If no preprocessor is defined, the Retriever renames the file's extension from `.raw` to `.log` and moves it to the `<LogQueue>/` directory.

6. The Loader takes the `.log` file and loads the data into the EDW using instructions contained in the PTL file. If the load succeeds, the Loader moves the `.log` file to the `<LogQueue>/done` directory.

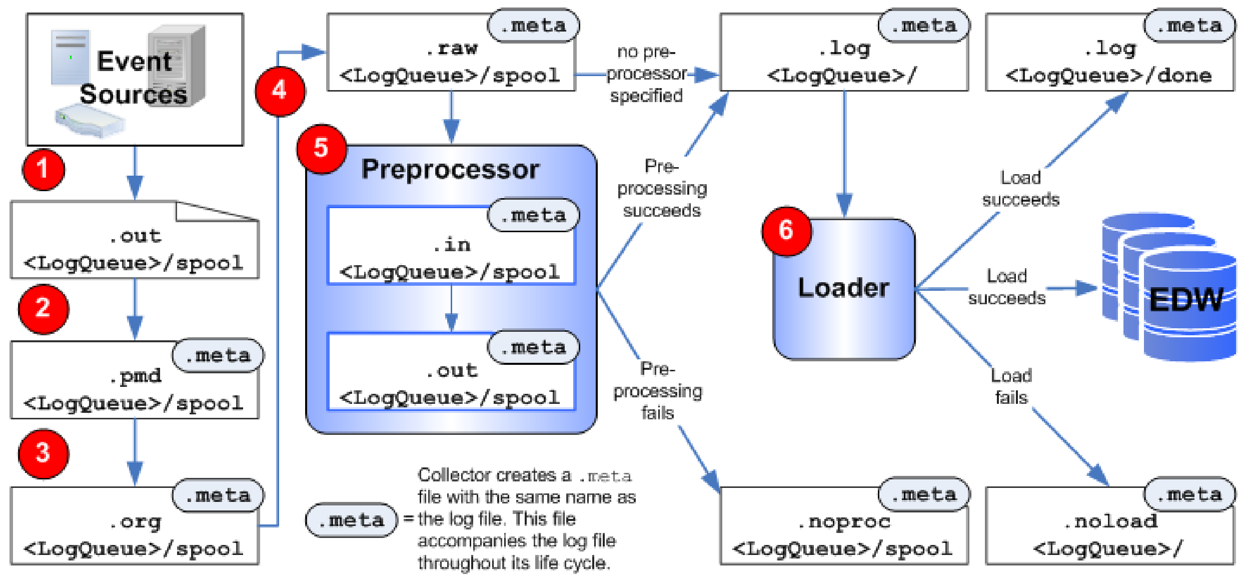
If the load fails, the Loader writes the log file to the `<LogQueue>/` directory with a `.noload` extension.

After debugging and correcting the failure, the HawkEye AP administrator should:

a Rename the file with a `.log` extension.

b Do not move or rename the `.meta` file.

Figure 2-1: Life Cycle of a Log File in a LogQueue



Event Sources
1
.out <LogQueue>/spool
2
.meta .pmd <LogQueue>/spool
3
.meta .org <LogQueue>/spool
4
.meta .raw <LogQueue>/spool
5
Preprocessor
.meta .in <LogQueue>/spool
.meta .out <LogQueue>/spool
no pre-processor specified
Pre-processing succeeds
Pre-processing fails
.meta .log <LogQueue>/
6
Loader
.meta .log <LogQueue>/done
Load succeeds
Load succeeds
Load fails
EDW
.meta .noproc <LogQueue>/spool
.meta .noload <LogQueue>/
.meta = Collector creates a .meta file with the same name as the log file. This file accompanies the log file throughout its life cycle.

Defining Retrievers

Retrievers are Collector modules that retrieve source log data and put it into files in log queues for loading into the EDW. As an option, retrievers can be configured to preprocess log files prior to loading.

You must configure at least one retriever in the `config.xml` file before the Collector can perform meaningful work. Retriever configurations are contained within `<Retriever ...></Retriever>` tags.

This section describes these topics:

- "Retriever Attributes", next
- "Retriever Elements", on page 28
- "Configuring Types of Retrievers", on page 32
- "Using Process Groups", on page 40

Retriever Attributes

A `<Retriever>` tag has these attributes, regardless of retriever type.

- **name**—(Required.) Unique name for the retriever
- **type**—(Required.) Type of retriever. For a complete list or receiver types, see "Configuring Types of Retrievers", on page 32.
- **enabled**—(Required.) Indicates whether the retriever is enabled (`1`) or disabled (`0`)

- **process**—(Optional.) Name of the process group in which to run the retriever. For more information, see "Using Process Groups", on page 40.

  TIP: Use this parameter to maintain a single version of the `config.xml` file on all of your HawkEye AP hosts.

- **ssl**—(Optional.) Use SSL; ignores invalid certs (`0` for disabled, `1` for enabled)

  ```
  <Retriever name="http1" type="http-push" enabled="1" ssl="1">
  ```

Example Retriever Attributes

The example below shows the definition of a File System retriever. Three attributes in the example are required for all retrievers: `name`, `type`, and `enabled`. The other attributes are used only with File System retrievers.

```
<Retriever name="myFileSystemRetriever" type="filesystem" enabled="1"
    method="hardlink" deleteOriginal="1" >
```

For more information on the attributes of specific types of retrievers, see "Configuring Types of Retrievers", on page 32.

Retriever Elements

In addition to attributes in the `<Retriever>` tag, a retriever configuration uses elements between the `<Retriever>` and `</Retriever>` tags. These are the elements that you can use:

- "LogQueue", on page 28
- "Preprocess", on page 30
- "RunOnHost", on page 30
- "SourceDir", on page 30
- "BackupDir", on page 30
- "SourceUser", on page 31
- "SourceServer", on page 31
- "SourceHost", on page 31
- "SourcePasswd", on page 31
- "SourceFile", on page 31
- "Listen", on page 31
- "ConfigFile", on page 31
- "Schedule", on page 31
- "PollInterval", on page 32
- "Period", on page 32
- "Plugin", on page 32

LogQueue

(Required by all retriever types.) The value specifies the log queue to which the retriever directs its output.

```
<LogQueue [id="n"|{high="n"|low="n"}]>log_queue_name</LogQueue>
```

You define the log queue referenced by *`log_queue_name`* in the `<LogQueues>` section of the configuration file. To learn how to define log queues, see “Configuring Log Queues”, on page 25.

You can load a log file to multiple EDW instances by configuring a single retriever with multiple log queues.

The `<LogQueue/>` element has these attributes.

- **id**—(Optional.) The order to use the log queues.

  ```
  <LogQueue id="1">local</LogQueue>
  <LogQueue id="2">remote</LogQueue>
  ```

- **high** | **low**—(Optional.) Allows you to pool log files into log queue based on the number from `0-99` that the Collector assigns them. For example, the following retriever divides all incoming web logs into 2 queues, `q1` and `q2`. It replicates those 2 queues to `q3` and `q4` respectively for backup clusters. The high of 49 and low of `50` specifies that the number of incoming logs should be split in half. That is, if a log file is assigned a value of `24`, a copy will be placed in queues `q1` and `q3`. If a subsequent log file is assigned the value `78`, a copy will be placed in queues `q2` and `q4`.

  ```
  <Retriever name="weblogs" type="filesystem" enabled="1"
      method="copy" deleteOriginal="1">
      <LogQueue high="49">q1</LogQueue>
      <LogQueue low="50">q2</LogQueue>
      <LogQueue high="49">q3</LogQueue>
      <LogQueue low="50">q4</LogQueue>
      <SourceFile>*.gz</SourceFile>
      <SourceDir>/path/to/log/files</SourceDir>
  </Retriever>
  ```

  A retriever places all log files that it handles in the queue if neither `high` nor `low` is specified.

  For example, if you add another queue without a high or low attribute to the example, half of the log files retrieved will go into `q1`, half into `q2`, with backups copied to `q3` and `q4`. All log files will also be copied to log queue `q5`.

  ```
  <Retriever name="weblogs" type="filesystem" enabled="1"
      method="copy" deleteOriginal="1" >
      <LogQueue low="50">q2</LogQueue>
      <LogQueue high="49">q1</LogQueue>
      <LogQueue high="49">q3</LogQueue>
      <LogQueue low="50">q4</LogQueue>
      <LogQueue>q5</LogQueue>
      <SourceFile>*.gz</SourceFile>
      <SourceDir>/path/to/log/files</SourceDir>
  </Retriever>
  ```

roundRobin

(Optional for all retriever types.)

"roundRobin=’1’"

Attribute indicates that log files are to be split on a line-by-line basis between two or more queues, instead of loaded as whole files. Use this function to distribute your data across multiple EDW clusters.

Example:

For each log file, split lines 50/50 between log queues "’a’ and ’b.’"

```
<Retriever name=’filesystem’ process=’files1’ enabled=’1’ deleteOriginal=’1’
type=’filesystem’ method=’hardlink’ roundRobin=’1’>
<RunOnHost>localhost</RunOnHost>
<LogQueue low="0" high="49">a</LogQueue>
<LogQueue low="50" high="99">b</LogQueue>
<SourceDir>/tmp/z</SourceDir>
<PollInterval>10 seconds</PollInterval></Retriever>
```

Preprocess

(Optional for all retriever types.) The values specify a program to run for preprocessing to occur.

```
<Preprocess [id="<n>"] type="{file|pipe}"
    [match="<regular-expression>"]>path-and-program</Preprocess>
```

For more information, see “Configuring Preprocessors”, on page 41.

RunOnHost

(Optional for all retriever types.) Specifies on which host to run the retriever. If not specified, the retriever runs on the local host where the `config.xml` file is located.

TIP: This attribute is useful if you are running multiple Collectors, as in daisy chains, which share a single `config.xml` file. The attribute specifies on which Collector host the retriever runs.

SourceDir

(Required for all retriever types, except LEA and HTTP Push retrievers.) File system directory from which to retrieve source log files.

IMPORTANT:

- DO NOT place files that you do not want collected in this directory. This will generate errors and the file may be deleted.
- DO NOT configure multiple retrievers to collect data from the same directory.

NOTE: If you specify a location on an NFS mount, make sure the retriever is in its own process group. For more information, see “Using Process Groups”, on page 40.

BackupDir

(Required only for LEA and HTTP Push Retrievers.) Backup locations to save backups of downloaded source log files.

```
<!-- Saves to filesystem path, in this case an NFS mount. -->
<BackupDir>file:///nfs1/backups</BackupDir>

<!-- Copies file with scp to example.com using fred's account. -->
<!-- Note: This requires public/private keypairs -->
<BackupDir>scp://fred@example.com/nfs2/backups</BackupDir>

<!-- Copies file with scp using -C for stream compression. -->
<BackupDir>scpc://fred@example.com/nfs2/backups</BackupDir>
```

SourceUser

Remote user to connect to via SSL, SFTP, FTP, or HTTP Get.

SourceServer

(Required for RDBMS retrievers only.) The database-specific portion of the DBI DSN.

SourceHost

Remote host; for example:

```
<SourceHost>10.0.1.230</SourceHost>
```

Host names can also contain a numeric range in {1-3} or ',' delimited list; for example:

```
<SourceHost>web{1-10}</SourceHost>
```

SourcePasswd

Used in FTP and HTTP Get Retrievers to specify the password for remote login from FTP and HTTP Get clients.

HTTP Push Retrievers can also specify a source password. The password is defined in the `collector_push` script using the option. For details, see “HTTP Push Retrievers”, on page 37.

SourceFile

(Required for all retriever types except RCP, SCP, and HawkEye Retriever types.)

<SourceFile>*.log</SourceFile>

The Collector translates the value of the `<SourceFile>` element to a regular expression. The Collector also uses this value to tell the Plugin which files can be accepted. See “RCP and SCP Retrievers”, on page 37, for an example.

Listen

For push port or ports and hosts; for security purposes, you can use this option to specify that Collector listen only on internal hosts.

```
!--  Remote Host:Port on which to listen -->
  <Listen>127.0.0.1:8001</Listen>
  <Listen>127.0.0.1:8003</Listen>

<!--  Local Port on which to Listen -->
   <Listen>8002</Listen>
```

ConfigFile

Specifies the absolute path of the HawkEye Retriever configuration properties file, which is named `agentless.prop`.

Schedule

Schedule for when you want the retriever to run. For more information, see “Scheduling Retrievers and Loaders”, on page 135 in the *Administration Guide.*

PollInterval

The `<PollInterval>` Interval defines how often, in seconds, the Retriever should run. See “Scheduling Retrievers and Loaders”, on page 135 in the *Administration Guide*.

Period

If `<PollInterval>` is used to define when the retriever runs, use `<Period>` to determine when the retrieving begins. See “The PollInterval and Period Elements”, on page 137 in the *Administration Guide*.

Plugin

Defines whether or not to retrieve a file.

You can define which files to accept and which files to ignore using regular expressions. For example:

```
<Plugin name="Default">
    <!--  ignore all files -->
    <File id="1" ai="ignore">.*</File>
    <!--  unless followed by a dot and at least one digit -->
    <File id="2" ai="accept">\.\d+$</File>
 </Plugin>
```

You can define how long source log files remain unchanged before a retriever transfers them for processing. For example,

```
<Plugin name="Default">
    <!-- File must remain unchanged for this long before we download -->
    <UnchangedFor>1 hour</UnchangedFor>
</Plugin>
```

For more information on using the `<UnchangedFor>` attribute, see See “Scheduling Retrievers and Loaders”, on page 135 in the *Administration Guide*.

IMPORTANT: The regular expression attempts to match the *file name* of files located in the directory specified by the `<SourceDir>` element of the Retriever. If a Preprocessor is also defined for the Retriever, note that the regular expression is compared against the *full URL* of the filename. For more information, see “Configuring Preprocessors”, on page 41.

Configuring Types of Retrievers

This section describes how to configure these types of retrievers.

- “File System Retrievers”, next
- “HawkEye Retriever on Windows”, on page 34
- “FTP Retrievers”, on page 35
- “SFTP Retrievers”, on page 36
- “RCP and SCP Retrievers”, on page 37
- “HTTP Get Retrievers”, on page 37
- “HTTP Push Retrievers”, on page 37
- “Script Retrievers”, on page 38

File System Retrievers

File System retrievers gather log files from specified file system locations where logs are placed by processes external to HawkEye AP. File system locations can be on local disks, NFS mounts, or Samba mounts. Local disks work best for File System retrievers. Each retriever gathers one log file.

NOTE: File system retrievers read the directory specified with the `SourceDir` element and go through the same process as noted under the topic "File Processing Performed by FTP Retrievers", on page 35.

Attributes for File System Retrievers

File System retrievers have these additional attributes in the `<Retriever>` tag:

- **method**—(required) the method of file-system access to source log files. Use one of these text values:
  - **copy**—Collects log data by using the Unix `cp` command.

    IMPORTANT: If used with an NFS mount, make sure the retriever belongs to its own process group. For more information, see "Using Process Groups", on page 40.
  - **hardlink**—Collects log data by making a hard link to the log data.

    IMPORTANT: Do not use `hardlink` when `SourceDir` is on an NFS or Samba mount.
  - **symlink**—Collects log data by making a symbolic link to the log data so that the actual bytes do not need to be moved. This link is then removed by the loader after loading completes. The original data is removed only if the retriever is configured to do so.

    IMPORTANT: Do not use `symlink` when `SourceDir` is on an NFS mount.
- **deleteOriginal**—(optional) specifies whether to delete original log files from the system after retrieval (`1`) or leave them (`0`). Do not use `deleteOriginal` if the method you specify is `symlink`.

Elements for File System Retrievers

File System retrievers use these elements:

- "RunOnHost", on page 30 (optional)
- "LogQueue", on page 28 (required)
- "SourceDir", on page 30 (required)

  NOTE: If you specify a location on an NFS mount, make sure the retriever is in its own process group. For more information, see "Using Process Groups", on page 40.

Example File System Retriever Configuration

The following examples show File System retrievers configured for different methods of file access.

File System Retriever with Copy as the Method

```
<Retriever name="fsbackup" type="filesystem" enabled="1"
   process="files1" method="copy" deleteOriginal="1" >
   <RunOnHost>host02.myco.com</RunOnHost>
   <LogQueue>primary</LogQueue>
   <SourceDir>/tmp/logs-fsbackup</SourceDir>
   <BackupDir>scp://joe@snoopy/local/home/joe/backups</BackupDir>
</Retriever>
```

File System Retriever with Hardlink as the Method

```
<Retriever name="fs" type="filesystem" enabled="1"
  process="files1" method="hardlink" deleteOriginal="1" >
  <RunOnHost>host01.myco.com</RunOnHost>
  <LogQueue>primary</LogQueue>
  <SourceDir>/tmp/logs-fs</SourceDir>
</Retriever>
```

File System Retriever with Symlink as the Method

```
<Retriever name="fssym" type="filesystem" enabled="1"
    process="files1" method="symlink" >
    <LogQueue>primary</LogQueue>
    <SourceDir>/tmp/logs-fssym</SourceDir>
</Retriever>
```

HawkEye Retriever on Windows

The HawkEye Retriever collects Windows event-log messages and application logs. It passes the collected data for batch processing by the Collector, the HawkEye Retriever can write to a Collector log queue that loads the data into the EDW. Figure 2-11 in the previous chapter describes and illustrates using the HawkEye Retriever both agentless and as an agent to collect the data and forward it to the Collector. When you configure the Retriever as an agent, you can collect Windows application data as well as event-log data from the Windows host.

IMPORTANT: When processing batch data, you first edit the Retriever configuration file to define properties specific to the objects that comprise the HawkEye Retriever. After you set the retriever properties, configure the Collector config.xml file to point to the properties file. For more information, see the following:

- configuring the retriever properties file—“Standard Configuration”, on page 61
- defining the retriever in config.xml—“Configuring the Retriever to Run Agentless”, on page 34 and “Example of Agentless HawkEye Retriever Configuration”, on page 34

Configuring the Retriever to Run Agentless

To define the Retriever to be agentless, you must use `config.xml` to define the following elements:

- “LogQueue”, on page 28 (required)
- “ConfigFile”, on page 31 (required)

  NOTE: If you specify a location on an NFS mount, make sure the retriever is in its own process group. For more information, see “Using Process Groups”, on page 40.

IMPORTANT: In addition to defining the properties for your HawkEye Retriever, you must configure authentication against a Windows server for agentless event-log retrieval. For more information, see “Configuring Authentication”, on page 103.

Example of Agentless HawkEye Retriever Configuration

The following example shows configuration of an agentless HawkEye Retriever in the `config.xml` file.

```
<Retriever name="windows1" type="windows" deleteOriginal="1" enabled="1">
   <RunOnHost>host02.myco.com</RunOnHost>
   <LogQueue>win_logqueue</LogQueue>
   <ConfigFile><HawkEye AP Home>/etc/collector/agentless.prop</ConfigFile>
</Retriever>
```

FTP Retrievers

FTP retrievers collect log data using the FTP protocol.

NOTE: FTP Retrievers transfer files only in ascii mode.

Attributes for FTP Retrievers

FTP retrievers have this additional attribute in the `<Retriever>` tag:

- **keepAlive**—(Optional.) Specifies whether to keep connections open between checks for new files (`1`) or to abandon connections (`0`).

Example FTP Retriever Configuration

```
<Retriever name="ftp001" type="ftp" enabled="1" process="ftp1"
   deleteOriginal="0" keepAlive="1" >
   <LogQueue>primary</LogQueue>
   <SourceDir>/local/home/demo/logs</SourceDir>
   <SourceHost>10.0.1.174</SourceHost>
   <SourceUser>hexis</SourceUser>
   <SourcePasswd>userdemo</SourcePasswd>
</Retriever>
```

IMPORTANT: FTP retrievers require a clear-text password in the `<SourcePasswd>` element. For security purposes, restrict access to the configuration file to root, the `hexis` user, and a limited number of users who modify the Collector configuration.

File Processing Performed by FTP Retrievers

FTP, SFTP, RCP, SCP, and File System retrievers connect to remote servers and request a directory listing of the `SourceDir`. The Collector keeps an internal database of filenames, modification dates, and byte sizes for every source log file. If a retriever finds new or modified files, it transfers them to the local disk for inspection.

The Default plugin can be used to further refine the list of files to be transferred. Configure the plugin to ignore patterns and/or enforce a *waiting period* on files. A waiting period prevents a file from being downloaded until it has stopped changing for a specified number of seconds. You could also write your own plugin with your own logic for determining which files should be transferred and which should be ignored. For example,

```
<Plugin name="Default">
  <UnchangedFor>10 minutes</UnchangedFor>
</Plugin><
```

For more information on using the `<UnchangedFor>` attribute, see “Scheduling Retrievers and Loaders”, on page 135 in the *Administration Guide.*

After transferring source log files, FTP retrievers perform an MD5 sum of each file and compare that value against known values of any file retrieved by the Collector. MD5 provides a very high level of assurance that the file has not been loaded before. Once the file passes this check, it is cleared for loading and put into the appropriate log queue. Its `MD5` is also added to the list of `MD5` s for future checks.

SFTP Retrievers

SFTP retrievers collect log data using the SFTP protocol.

NOTE: Collector does not work with Windows versions of SFTP. To ensure compatibility, use a version of SFTP that is included in the versions of Linux that the Collector can run on.

Attributes for SFTP Retrievers

SFTP retrievers have this additional attribute in the <Retriever> tag:

- **keepAlive**—(Optional.) Specifies whether to keep connections open between checks for new files (`1`) or to abandon connections (`0`).

Establishing Access to Desired Data on Target Server

Unlike the FTP retriever, an SFTP retriever does not have the <`SourcePasswd`> tag. To enable the SFTP retriever to get data from the target server, you must set up SSH (Secure Shell) trust between the HawkEye AP system user on the server hosting the Collector and the user on the target server, which you specify in the <`SourceUser`> tag for the retriever. The user on the target server must have access to the desired data.

Set up trust by copying the contents of the HawkEye AP user's public key file on the server hosting the Collector into the authorized key files of the user on the target server. Follow these steps:

1. On the Collector host, change to the HawkEye AP user:

```
su -l hexis
```

2. Copy the public key file to the target user on the target host.

```
scp ~/.ssh/id_dsa.pub <target_user>@<target_host>:~
```

3. On the target host, change to the target user.

```
su -l <taget user>
```

4. Copy the contents of the HawkEye AP user's public key file into the target user's authorized key files.

```
cat ~/id_dsa.pub>> ~/.ssh/authorized_keys2
```

5. Delete the public key file.

```
rm ~/id_dsa.pub
```

Example SFTP Retriever Configuration

```
<Retriever name="sftp1" type="sftp" enabled="1" process="ftpGroup"
    deleteOriginal="0" keepAlive="1" debug="1">
    <LogQueue>remote</LogQueue>
    <SourceDir>queue</SourceDir>
    <SourceHost>localhost</SourceHost>
    <SourceUser>hexis</SourceUser>
    <BackupDir>file:///tmp/backup1</BackupDir>
    <Preprocess id="2" type="pipe" match="mail">/usr/bin/wc</Preprocess>
    <Preprocess id="1" type="file" match="apache">/bin/cp</Preprocess>
</Retriever>
```

RCP and SCP Retrievers

RCP and SCP retrievers collect log data using a list of file names and/or wildcards. For example:

```
<SourceFile>*.log</SourceFile>
```

Elements for RCP and SCP Retrievers

You can configure RCP and SCP retrievers to ignore files with a wildcard expression. Enclose the expression in a `<Plugin>` element, as the next example shows:

```
<Plugin name="Default">
    <File ai="ignore">~$</File>
</Plugin>
```

Usage Notes for RCP and SCP Retrievers

- Hexis Cyber Solutions does not recommend the use of RCP or SCP if you are transferring large log files, because the wildcard selection may result in repeated large file downloads.
- RCP and SCP retrievers make a connection to the remote server and download whatever source log files are specified in the `<SourceFile>` elements. Files may be specified by individual filename or by wildcard specification. The retrievers then perform MD5 sums on each file and follow the same process as FTP retrievers for loading files. For more information, see "File Processing Performed by FTP Retrievers", on page 35.

HTTP Get Retrievers

HTTP Get retrievers gather log files specified by filenames and/or wildcard expressions. To avoid the wildcard issue mentioned above (where large files could be repeatedly downloaded due to the broadness of a wildcard match), this type checks the file size and date modified (when available) to determine whether or not to download files.

```
<Retriever name="http-get" type="http-get" enabled="1" process="1" ssl="1">
    <LogQueue>local</LogQueue>
    <SourceHost>127.0.0.1:80</SourceHost>
    <SourceDir>/~demo</SourceDir>
</Retriever>
```

Usage Notes for HTTP Get Retrievers

- The source directory must have indexing enabled, as set by your web server.
- If an entire directory is specified, and the server supports directory listings, the HTTP Get Retriever parses the directory listing for modification date and byte size, and goes through the same process as FTP retrievers for loading files. For more information, see "File Processing Performed by FTP Retrievers", on page 35.

HTTP Push Retrievers

HTTP push retrievers accept multiple, simultaneous, incoming HTTP connections under a single process. This type requires the use of a remote sender script or program to send files. On Unix, the sender is the `collecotor_push` script; on Windows the sender is the `collector_push.exe` program.

Example HTTP Push Retriever Configuration

```
<Retriever name="http-push" type="http-push" enabled="1" ssl="0">
    <LogQueue>local</LogQueue>
    <Listen>127.0.0.1:8001</Listen>
    <SourceUser>demo</SourceUser>
    <SourcePasswd>mypasswd</SourcePasswd>
</Retriever>
```

Configuring the Remote Sender for HTTP Push Retrievers

The `collector_push` script for Unix systems and the `collector_push.exe` for Windows run on the log-file host. Run the script or program from the command line, or set up a recurring job on the remote host or server.

The remote sender takes the following options:

- **--source**—directory to scan for files
- **--dest**—`<host>:<port>` for Collector
- **--name**—unique name for client (can be anything, but usually the name of the host where log's data came from)
- **--ssl**—`0` for enabled, `1` for disabled
- **--user**—authenticates to the value set in the `<SourceUser>` element
- **--passwd**—password sent to the HTTP Push retriever for authentication
- **--delete**—deletes files after successful load run
- **--delay**—can be used with `--delete`

For example:

```
collector_push --source="localdir" --dest ="collector.host.com:9999"
```

Script Retrievers

A script retriever allows you to collect data using a script or other program that you or another developer creates. You can use the script to perform any type of data collection or generation. For example, you could write a script to query an RDBMS and send the output to the Collector, or the script could execute a command such as `lastlog` whose output is captured and sent to the Collector.

The script can also add additional metadata to the retrieved data prior to loading. The script produces an output file that is picked up by the Collector and loaded using standard loading procedures.

A script or program used with the script retriever must be executable from the command line and must direct output to a file whose name is provided by a command-line argument.

Script Retriever Elements

- `Retriever`–This is the containing element for each retriever. To create a script retriever, set the `type` attribute to "`filescript`". For example:

  ```
  <Retriever
    type="filescript"
  ```

```
  ...
</Retriever>
```

- `RunOnHost`—(Optional.) A designated host on which the retriever runs. Without this element, the retriever runs on the host where the `config.xml` file is located. See "RunOnHost", on page 30.
- `LogQueue`—See "LogQueue", on page 28
- `SourceEnv`—Specify environment variables and their values that are set prior to executing the script. Specify the variable's name with the `name` attribute, and its value within the element. You can specify multiple `SourceEnv` elements to define additional environment variables. For example:

  ```
  <SourceEnv name="user">john</SourceEnv>
  <SourceEnv name="hostname">myhost.com</SourceEnv>
  ```

- `SourceCommand`—Specify the command line used to run the script. The optional `multiple` attribute has two possible values:
  - "`1`"—Execution of the command is repeated until an empty output file is produced.
  - "`0`" (or not specified)—The command executes only once for each interval specified with the `PollInterval` element.

  Include any command line arguments for the script or program as part of the command line within the `SourceCommand` element.

  Use "`%F`" to represent the argument to your script or program that specifies the output file name. The script must accept at least a single input argument which represents the filename where the script writes its output. The Script retriever supplies the filename when the command executes. For example:

  ```
  <SourceCommand multiple="1">/opt/myScript %F</SourceCommand>
  ```

- `PollInterval`—(Recommended.) Specify how often, in seconds, the script should run. For example:

  ```
  <PollInterval>3600</PollInterval>
  ```

Writing a Script for Use with the Script Retriever

The script retriever runs the command specified with the `<SourceCommand>` element. The command referenced by this element may be a script or compiled program, created by a developer, or provided by HawkEye AP.

IMPORTANT: The script runs as the same user as the Collector and can perform many potentially destructive actions such as deleting or re-writing files on the file system. Make sure your script only performs the expected data collection activities. Note also that a script that generates excessive errors can flood the Collector's log file with error messages.

(Optional) You can also specify metadata in the script to describe the file's source and modification time. You add this metadata by writing instructions in the script that write a metadata file whose name is the same as the output file but ends with "`.meta`". This file should contain either or both of the following two lines:

```
FileSource: <name of source file>

FileModTime: <number of seconds since epoch (as an integer)>
```

The `FileSource` metadata is used by the Collector instead of the actual filename when matching to a filename specified by the expression in the `match` attribute in the `<Preprocess>`, `<Postprocess>` and `<PTL>` elements.

NOTE: Format the lines exactly as shown above, substituting the correct values after the colon and space characters.

Sample Script Retriever Configuration

```
<Retriever
  name="filescriptExample"
  type="filescript"
  process="files1"
  enabled="1">
  <RunOnHost>localhost</RunOnHost>
  <LogQueue>local</LogQueue>
  <SourceEnv name="user">john</SourceEnv>
  <SourceCommand multiple="1">/opt/myScript %F</SourceCommand>
  <PollInterval>3600</PollInterval>
</Retriever>
```

Sample Script

The following simplified sample script writes the date to an output file represented by the `$1` parameter and also writes metadata to a file whose name begins with the file represented by the `$1` parameter and ends with "`.meta`". When this script is referenced by the Script retriever sample shown above, the Script retriever generates a filename and substitutes it for the `%F` parameter you specify in the `<SourceCommand>` element.

```
#!/bin/bash
#myScript.sh
date>$1
echo "FileSource: myFileSource">$1.meta
```

STDERR, STDOUT, and Exit Status

When the script executes:

- `STDERR` output from the script is logged as an error in the Collector’s log.
- `STDOUT` is logged as verbose in the Collector’s log.
- Non-zero exit status is logged as an error in the Collector’s log and the output file is discarded.
- Zero exit status with an empty output file means that there is no data available to load.
- Zero exit status with no output file is logged as an error in the Collector’s log.

Using Process Groups

Process groups reduce the system processes required to run retrievers compared to running retrievers individually. Specify which process group a retriever belongs to with the `process` attribute in the `<Retriever>` tag.

Depending on their type, retrievers are *blocking* or *non-blocking*. Blocking retrievers that are scheduled to run at the same time are queued up in a first-come, first-served basis. They run sequentially, one after the other. Non-blocking retrievers that are scheduled to run at the same time run in parallel. File system and FTP retrievers are blocking. All other types are non-blocking.

When you define a retriever, you can specify which process group the retriever is part of. Consider a system with a large number of retrievers. If all of the retrievers start at one time, the system becomes overwhelmed with processes. Group blocking retrievers, such as file system and FTP retrievers, in a process group separate from non-blocking receivers. This ensures that non-blocking retrievers run in parallel and are not interrupted by the sequential processing of a blocking receiver.

TIP: For improved efficiency, do not mix blocking and non-blocking retrievers in the same group.

IMPORTANT: Place file system retrievers that access NFS mounts in their own process groups. In the following example, `nfs1` is the process group. Other retrievers that access the same NFS mount should also specify the process group `nfs1`.

```
<Retriever name="fscp" type="filesystem" process="nfs1" enabled="1"
   method="cp" >
   <LogQueue>primary</LogQueue>
   <SourceDir>/tmp/logs-fssym</SourceDir>
</Retriever>
```

You can also define a process group to manage retrievers that gather log files from many machines that generate very small logs.

Configuring Preprocessors

Retrievers can have one or more preprocessing scripts identified to transform log data prior to writing log files to log queues. Examples of preprocessing include concatenating multi-line log entries into single-line entries, replicating data among lines, and stripping out unwanted characters, such as nulls.

Configure preprocessors in `<Retriever/>` sections of the `config.xml` file, using the following example syntax:

```
<Retriever name="fssyslog" type="filesystem" process="files1" method="copy"
 deleteOriginal="0" enabled="0">
    <LogQueue>syslog_lq</LogQueue>
    <SourceDir>/home/sensage/logs/syslog</SourceDir>
    <Preprocess id="2" type="pipe">/usr/bin/wc</Preprocess>
    <Preprocess id="1" type="file" match="apache">/usr/bin/procsf</Preprocess>
</Retriever>
```

Use a `<Preprocess/>` element within a `<Retriever>` section to specify a script or binary to run on the log files the retriever collects prior to loading.

IMPORTANT: Preproccess programs must be written to handle the character encoding of the source log files that the retriever collects. For more information, see "Configuring Log Queues", on page 25.

The `<Preprocess>` element has the following attributes:

- **id**—Specifies the order in which preprocessors are applied for this retriever. Specify `1` for the preprocessor that you want the Retriever to run first, `2` for the second, and so on.
- **type**—(Required.) Determines how Collector sends data to the preprocessor; specify one of the following values:
  - **file**—a script to which two command-line parameters are passed: the original filename, and the "save to" filename. For example: `/usr/bin/procsf/test.preproc $infile $outfile`

  - **pipe**—handles streaming data through a pipe. For example:cat $infile | /usr/bin/wc/test.preproc > $outfile

  NOTE: The values **'**pipe' and 'file' represent two different values that you can assign to 'type'. Collector replaces these values.

- **match**—(Optional.) A regular expression that describes which matching log files will be run through this preprocessor. In the following example, `mail` specifies that the preprocessor operates on log files with “mail” in the filename.

```
<Preprocess id="2" type="pipe" match="mail">/usrbin/wc</Preprocess>
<Preprocess id="1" type="file" match="apache">/bin/cp</Preprocess>
```

  NOTE: When matching file names using the `match` attribute of the `<Preprocessor>` element, note that the regular expression is compared against the *full URL* of the filename, for example:

```
  file://local/incoming/myFilename.loadme
```

  If you also define a Plug-in for the same retriever, note that the regular expression defined in a `<File>` element matches only against the *file name portion of the URL*, and assumes the file is located in the directory specified by the `<SourceDir>` element of the retriever. For example,

```
  myFilename.loadme
```

  To correctly load data from multiple retrievers that also use Plug-ins, make sure to construct the regular expressions in the `<File>` and `<Preprocess>` elements correctly. For more information, see “Plugin”, on page 32.

DEFINING LOADERS

Loaders are Collector modules that load data into the EDW. You can define one or more loaders in a single Collector instance within the `config.xml` file. In general, create a different loader for each EDW instance into which you want to load data.

Within a loader configuration, you can specify different PTL files to handle different event-log types. For example, one PTL file handles firewall event logs and another PTL file handles syslog event logs. PTL files specify how log entries are parsed and loaded by the EDW into target schemas during loads.

Specify loaders within the `<Loaders>` section of the `config.xml`, using the following example:

```
<Loaders>
  <Loader name="fs" enabled="1" trickle="1" >
    <LogQueue>local</LogQueue>
    <PTL table="web" type="file" enabled="1">/ptls/apache_access-1BT.ptl</PTL>
    <SLSInstance>slshost.myco.com:2112</SLSInstance>
    <SLSUser>administrator</SLSUser>
    <SLSPasswd>changeme</SLSPasswd>
    <ParseFailures>
      <AbortOnFailure>0</AbortOnFailure>
      <LogFile><HawkEye AP Home>/var/log/collector/apache-parsefailures.log</
LogFile>
    </ParseFailures>
  </Loader>
</Loaders>
```

Loader Attributes

The `<Loader>` tag has these attributes:

- **name**—(Required.) Unique name for the loader.
- **enabled**—(Required.) `0` for disabled, or `1` for enabled.
- **trickle**—(Optional.) `0` for disabled, or `1` for enabled. The default is 0. Treats the load as a trickle feed (either small loads or an unsorted data stream).

For example:

```
<Loader name="fs" enabled="1" trickle="1">
```

In the following example the trickle argument is added to the Collector configuration by setting the trickle attribute to ’1’ and the table is compacted by setting the compact attribute to 1.

```
<Loader enabled=’1’ trickle=’1’ name=’loader 1’>
  <RunOnHost>localhost</RunOnHost>
  <LogQueue>local</LogQueue>
  <PTLtable=’sometable’namespace=’demo’
  enabled=’1’ compact=’1’>/tmp/someptl.ptl</PTL
  <PollInterval>1 minute</PollInterval>
  <SLSInstanceid=’1’>127.0.0.1:%SLS_PORT%</SLSInstance>
  <SLSUSER>administrator</SLSUser>
  <SLSSharedKey>%SECRET%<SLSSharedKey>
  </Loader>
```

NOTE: With EDW Trickle Feed Load (TFL) support, data loaded with TFL option needs to be compacted periodically so that the EDW data structures stay small and do not cause unnecessary delays when a manual compact operation is invoked after large amount of data is accumulated in TFL data structures.

Therefore, Collector can enable EDW Compact operation in a timely manner so that the data loaded with EDW Trickle Feed option is merged into EDW data leaves

Compaction parameters would be defined within a <Loader> section:

- Compaction schedule. and /or
- A threshold determined by Rows and/or Bytes and/or Load Counts

You can use more than one loader against a given table.

- Compaction threshold statistics would be stored globally (within a given collector) on a table by table basis.
- Compaction would lock out loads to a table globally (within given collector).

The following <Compact> attributes support the above:

- `loadsThreshold` - Number of loads after which to trigger a compact (default – no trigger on this)
- `rowsThreshold` - Numbers of rows after which to trigger a compact (default – no trigger on this)
- `megsThreshold` - Number of mega bytes after which to trigger a compact (default – no trigger on this)

If you choose none of the above options, Collector uses these parameters for scheduling:

```
<!-- simple parameterized compact ->
<Compact loadsThreshold="100" rowsThreshold="100000000"
megsThreshold="1000000000"/>
<!-- one parameter only, compact only at size threshold ->
<Compact megsThreshold="1000"/>
<!-- compact only at one minute after midnight - Schedule>
<Compact> <Schedule>1 0 * * *</Schedule> </Compact>
<!-- always compact at one minute after midnight, or after 10 gigs is loaded
since last compact -->
<Compact megsThreshold="10000">

    <Schedule>1 0 * * *</Schedule>

    </Compact>
```

Loader Elements

In addition to attributes in the `<Loader>` tag, a loader configuration uses these elements between the `<Loader>` and `</Loader>` tags:

- “LogQueue”, next
- “PTL”, on page 45
- “Merge”, on page 46
- “SLSInstance”, on page 48
- “SLSUser”, on page 48
- “SLSPasswd”, on page 48
- “SLSSharedKey”, on page 48
- “Schedule”, on page 49
- “Schedule”, on page 49
- “PollInterval”, on page 49
- “Period”, on page 49
- “Compact”, on page 49
- “DaisyChainHost”, on page 50

Description of the Loader Elements

LogQueue

(Required.) The value specifies the log queue from which the loader draws its input.

<LogQueue>*log_queue_name*</LogQueue>

You define the log queue referenced by *`log_queue_name`* in the `<LogQueues>` section of the configuration file. To learn how to define log queues, see “Configuring Log Queues”, on page 25.

PTL

(Required.) The `<PTL>` section of a loader configuration has two formats:

- “Simple Format for Specifying PTL Files”, on page 45
- “Extended format for Specifying PTL Files”, on page 45

Simple Format for Specifying PTL Files

In the simple format, the PTL section is a simple element. The value specifies which PTL file to use with the loader. The `<PTL>` element takes the following attributes:

- **id**—(Optional.) Priority; `1` is processed first, `2` next, and so on (recommended)
- **namespace**—(Optional.) The default is `default`, or whatever namespace your datastore accesses (recommended).
- **table**—(Required.) Table the data will get loaded into.
- **type**—(Optional.) Must be `file`; default is `file`.
- **match**—(Optional.) Regular expression that matches incoming log file names; if not included, will look for all available log files.
- **enabled**—(Required.) `0` for disabled, `1` for enabled.
- **trickle**—(Optional.) `0` for disabled, or `1` for enabled. The default is 0. Treats the load as a trickle feed (either small loads or an unsorted data stream).

  Note that if the <PTL...> element contains the optional ’compact’ attribute, the table is only compacted if the attribute is set to ’1’. For example:

  ```
  <PTL table =’xyzzy2’ namespace=’d’ enabled=’1’ compact=’1’>/tmp/sap.ptl</PTL>
  ```

- **compact**—(Optional.) `0` for disabled, or `1` for enabled. The default is 0. Compacts the specified table.

Extended format for Specifying PTL Files

In the extended format, the PTL section takes elements, including a `<Location>` element that specifies the PTL file to use with the loader. The extended format takes the same attributes as the simple format. The extended format also takes these elements:

- **Location**—The `<Location>` element specifies the PTL file to use.
- **LoadOption**—The `<LoadOption>` section specifies command-line options to pass to the `atload` command that the loader runs. One of the load options, `--override`, is useful to override time zones, `MD5` (checksum) values, server names, or other values that are constant for an entire load.

  For additional load options that control how the `atload` command functions, see Loading Data into the EDW in Chapter 3: Loading, Querying, and Managing the EDW in the *Administration Guide*.

- **ParseFailures**—The ParseFailures section specified within the PTL element overrides the default parse failures section given for the particular loader.

Example Loader Configuration

If you need to specify extra `atload` command-line arguments, use the following syntax:

```
<PTL table="main" namespace="myNamespace" type="file" match="error_" trickle="1"
compact="1">
   <Location>apache_error-1BT.ptl</Location>
   <LoadOption>--override=TZ:'GMT'</LoadOption>
   <ParseFailures>
        <AbortOnFailure>1</AbortOnFailure>
        <LogFile><HawkEye AP Home>/var/log/collector/lea-parsefailures.log</
LogFile>
   </ParseFailures>
</PTL>
```

In the example above, the load is treated as trickle feed and the table is compacted. The `<LoadOption>` element specifies a macro-override value for the expression macro `time zone`, which is defined in the PTL file `apache_error-1BT.ptl`. The override sets the time zone to GMT. The `<ParseFailures section>` specifies overrides for default parse-failure settings.

International Support in PTL Files

PTL files must be written with UTF-8 encoding. This lets the regular expression match international characters in source log entries. Source log files can be written with a variety of encoding schemes, but log entries are converted to UTF-8 encoding before evaluation by the regular expression in the PTL file.

Merge

(Optional.) The `<Merge>` section of a loader configuration is useful when the log queue fills with many small files. Too many such files can cause fragmentation and can severely impact performance. The Merge section allows you to configure postprocessing and merging and/or sorting of the incoming log files prior to loading into the EDW.

NOTE: A loader configured with the Merge option should not also specify a PTL. The Merge loader passes the data it merges to another loader that performs the actual loading into the EDW. The second loader specifies the PTL. The Merge loader specifies only the log queue and the merge processing.

The `<Merge>` section takes the following attributes:

- **loader**—(Required.) Specifies the destination loader; after the merge completes, the merged file is delivered to this loader for loading.
- **enabled**—(Required.) `0` for disabled, `1` for enabled.
- **maxMB**—(Required.) Specifies the total size of all available files before the merge occurs; in other words, merge occurs when all available files exceed specified size.
- **age**—(Required.) Specifies the maximum age of the oldest available file before the merge occurs; in other words, merge occurs when the oldest available files exceed specified age.
- **minAge**—(Recommended.) Specifies the minimum age of the oldest available file before the merge occurs; in other words, merge waits until the oldest available files exceeds specified age.
- **compress**—(Required.) `0` for false, `1` for true. If set to true, the Collector uses the `gzip` compression utility on the resulting merged file, which provides the file with a `.gz` extension. This attribute is useful when running over a WAN.

- **sort**—(Required.) `0` for false, `1` for true. If true, the Collector runs the Unix sort command on the resulting merged file.
- **sortOptions**—(Optional.) When you enable sorting, you can specify an alternative directory for temporary files. The `/tmp` directory is typically used but often has insufficient space. Specify this option in a quoted string that begins with `-T`; for example: `"-T /tmp/s"`.

The `<Merge>` section takes the following elements:

- **Postprocess**—(Required.) Specify the name of the postprocessor and, optionally, command-line parameters.

  Merge loaders can have one or more postprocessing scripts identified to transform log data prior to passing the data to a loader that actually writes the log files to the EDW.

  NOTE: Merging does not always require a postprocessing script. For example, postprocessing isn't necessary if the Loader is only sorting the data. In this case, you can pass the data unchanged through the log source by including the `cat` command in the postprocess element. The following illustrates how you might use this command:

  ```
  <Postprocess id='1'>(cat)</Postprocess>
  ```

  This element takes the following attributes:

  - **id**—(Optional.) Specifies the order in which preprocessors are applied for this Loader. Specify `1` for the preprocessor that you want the Loader to run first, `2` for the second, and so on.
  - **match**—(Optional.) A regular expression that describes which matching log files will be run through this postprocessor. In the following example, `mail` specifies that the postprocessor operates on log files with "mail" in the filename.

    ```
    <Postprocess id="1" match="snail">parseSnail.pl EST</Postprocess>
    <Postprocess id="2" match="mail">(gunzip | parseMail.pl %F)</Postprocess>
    ```

  NOTE: You can prepend a value to the postprocessed output by specifying the value after the name of the postprocessor. For example:

  - Specify the time zone:

    ```
    <Postprocess id="1" match="snail">parseSnail.pl EST</Postprocess>
    ```

  - Specify the special `%F` variable that captures the file name:

    ```
    <Postprocess id="3" match="mail">(gunzip | parseMail.pl %F)</Postprocess>
    ```

    The `parseWin.pl` Perl script above gets data from standard input and writes it to standard output. The Collector replaces `%F` is with the original file name, which you can use to influence processing. For example you can use the original file name to determine the file's timestamp.

  IMPORTANT: If the postprocessor output is to be sorted, prepend the epoch time to its output. If you prepend a timestamp or your postprocessor makes other changes to the file, you must modify the PTL in the associated loader to handle the changes.

- **PollInterval**—(Recommended.) Specify how often the data should be merged. Typically you set this value to the same value as the `age` attribute. Set the value in seconds.

Example Loader Configuration

The example below illustrates how you can merge multiple small files before loading them into the EDW:

```
<LogQueue>mergeQ</LogQueue>
<Merge loader="mail" enabled="1" maxMB="1000" age="2 hours" minAge="1 hour"
compress="1" sort="1" sortOptions="-T /tmp/s">
   <Postprocess id="1" match="mail">parseMail.pl %F</Postprocess>
   <Postprocess id="3" match="snail">parseSnail.pl EST</Postprocess>
</Merge>
<Merge loader="syslog" enabled="1" maxMB="5000" age="3 hours" minAge="1 hour"
compress="0" sort="0">
   <Postprocess id="2" match="syslog">parseSyslog.pl</Postprocess>
   <Postprocess id="4">parseRemainder.pl</Postprocess>
</Merge>
<PollInterval>3600</PollInterval>
```

The example above illustrates two Merge elements that load data from the same log queue. Each Merge element contains two postprocessor elements, each with its own ID attribute. The ID values specify that one postprocessor in each Merge element is processed before the processing of the second postprocessor in each Merge element.

The first Merge element specifies compression and sort. The second does not.

SLSInstance

(Required.) Host name and port number of the EDW instance to load data into. For example:

```
<SLSInstance>slshost.myco.com:2112</SLSInstance>
```

Remember that you can run multiple instances of EDW on a single host, and that each instance is identified by a different port number.

SLSUser

(Optional.) The user name to talk to the EDW. The default is `administrator`.

SLSPasswd

(Optional.) The password for the EDW user. The default is `changeme`. You can specify a shared key instead of an SLS password.

For more information about the EDW user and password or shared keys, see the HawkEye AP Analyzer Documentation.

SLSSharedKey

A "shared secret" password used to authenticate the loader. Use this in place of the `SLSPasswd`. The value of the shared secret is contained in the following file:

```
<HawkEye AP Home>/etc/sls/instance/<instance-name>/shared_secret.asc
```

For example:

```
  <Loader enabled='1' name='l_hp_skm_audit_batch'>
       <RunOnHost>localhost</RunOnHost>
       <LogQueue>q_hp_skm_audit_batch</LogQueue>
```

```
    <SLSInstance>sls01.east.myco:8072</SLSInstance>
    <SLSUser>administrator</SLSUser>
    <SLSSharedKey>e45ede21c4b81a2ac0c8417ea6846551</SLSSharedKey>
    <PTL table='hp_skm_audit_batch' namespace='analytics' type='file'>
      <Location>/opt/sensage/analytics/adapters/hp_skm_audit_batch/
        hp_skm_audit_batch.ptl</Location>
      <LoadOption>--override=TZ:'America/Los_Angeles'</LoadOption>
    </PTL>
  </Loader>
```

Schedule

Times that you want the loader to run. You can include several `<Schedule>` elements. For example, one schedule specifies hours for week days and another schedule specifies hours for week-ends.

For more information on how to specify a schedule, “Scheduling Retrievers and Loaders”, on page 135 in the *Administration Guide.*

PollInterval

The `<PollInterval>` Interval defines how often, in seconds, the Loader should run. See “The PollInterval and Period Elements”, on page 137 in the *Administration Guide*.

Period

If `<PollInterval>` is used to define when the Loader runs, use `<Period>` to determine when the loading begins. see “The PollInterval and Period Elements”, on page 137 in the *Administration Guide*.

Compact

(Optional.) Schedule the loader to compact data incrementally. This element can be used with the trickle attribute. See “Loader Attributes”, on page 43.

The `<Compact>` element takes the following attributes:

- **timeout**—(Required.) Specifies the maximum amount of time for the compacting.
- **archived**—(Optional.) `0` for disabled, `1` for enabled. If set to true, the Collector compacts archived as well as local data.

The `<Compact>` element takes the following element:

- **Schedule**—(Required.) Specifies the time the compacting occurs. For more information on how to specify a schedule, “Scheduling Retrievers and Loaders”, on page 135 in the *Administration Guide.*

The following example configures the Collector to run a compact at midnight for up to 2 hours. It compacts archived as well as local data.

```
<Loader>
...
   <Compact timeout="2 hours" archived="1">
      <Schedule> * 0 * * *</Schedule>
   </Compact>
</Loader>
```

Example 2 configures the trickle attribute setting to ’1’ and the table is compacted by setting the compact attribute to ’1’.

```
<Loader enabled=’1’ trickle=’1’ name=’loader 1’>
  <RunOnHost>localhost</RunOnHost>
  <LogQueue>local</LogQueue>
  <PTLtable=’sometable’namespace=’demo’
  enabled=’1’ compact=’1’>/tmp/someptl.ptl</PTL
  <PollInterval>1 minute</PollInterval>
  <SLSInstanceid=’1’>127.0.0.1:%SLS_PORT%</SLSInstance>
  <SLSUSER>administrator</SLSUser>
  <SLSSharedKey>%SECRET%<SLSSharedKey>
  </Loader>
```

DaisyChainHost

(Optional.) Links daisy-chain loaders to daisy-chain retrievers; tars up metadata files and log files; pair with daisy-chain preprocessor and daisy-chain retriever. For more information, see “Creating Daisy Chains”, on page 51.

Example Loader Configurations

The following examples show several types of loaders.

Example File System Loader Configuration

```
<Loaders>
   <Loader name="fs" enabled="1">
       <LogQueue>local</LogQueue>
        <PTL table="web" type="file" enabled="1">/usr/apache_access-1BT.ptl</PTL>
       <SLSInstance>slshost.myco.com:2112</SLSInstance>
       <ParseFailures>
           <AbortOnFailure>0</AbortOnFailure>
           <LogFile><HawkEye AP Home>/var/log/collector/apache-
parsefailures.log</LogFile>
       </ParseFailures>
  </Loader>
</Loaders>
```

Example Firewall Loader Configuration

```
<Loaders>
   <Loader name="firewalls" enabled="1">
      <LogQueue>remote</LogQueue>
      <PTL table="main" namespace="hq" type="file”
          match="access_">apache_access-1BT.ptl</PTL>
      <SLSInstance>slshost.myco.com:2112</SLSInstance>
      <Schedule>* 0-5 * * mon-fri</Schedule>
      <Schedule>* * * * sat-sun</Schedule>
      <ParseFailures>
         <AbortOnFailure>1</AbortOnFailure>
         <LogFile><HawkEye AP Home>/var/log/collector/lea-parsefailures.log</
LogFile>
      </ParseFailures>
    </Loader>
</Loaders>
```

Daisy-Chain Loader Example

```
<Loaders>
    <Loader name="daisychain" enabled="1">
        <LogQueue>daisychain</LogQueue>
        <DaisyChainDir>/tmp/daisychain</DaisyChainDir>
    </Loader>
</Loaders>
```

For more information on setting up daisy chains, see “Creating Daisy Chains”, on page 51.

COLLECTOR CONFIGURATIONS FOR NETWORKING

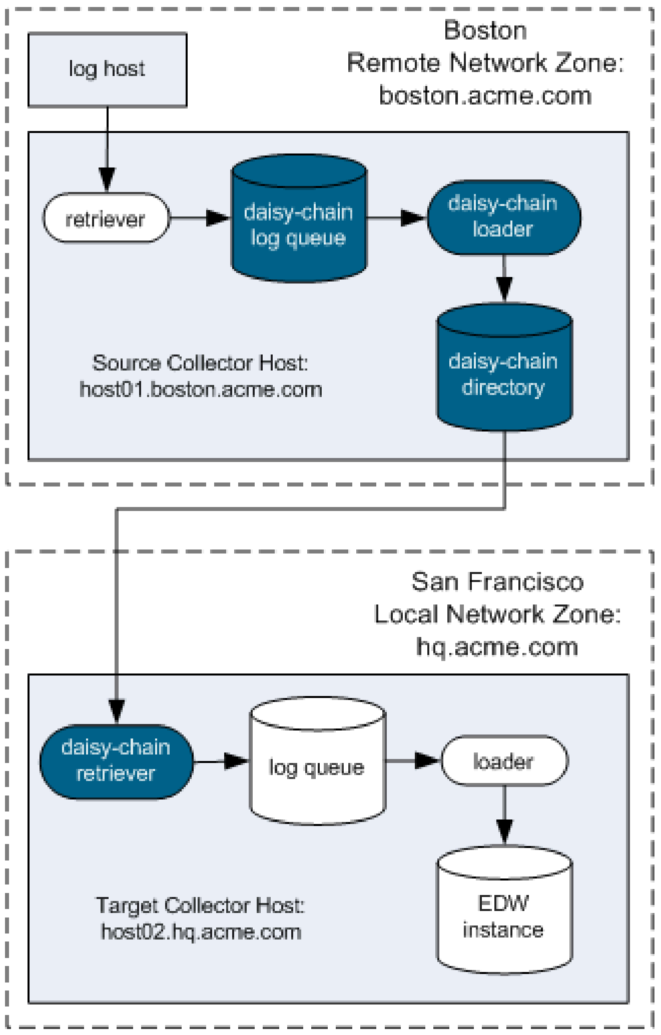
This section provides information when you are working with network zones that may require special Collector configurations. The topics discussed are:

- Daisy Chaining to send event-log data to a Central Collector from a remote Collector. See “Creating Daisy Chains”, next.

Creating Daisy Chains

Daisy chains are special Collector configurations that allow the output of one Collector instance to serve as the input to another Collector instance. This configuration provides for scalability and can overcome obstacles that network zones impose.

For example, you have one network zone for the San Francisco headquarters (hq.acme.com) and another network zone for the Boston field office (boston.acme.com). The EDW for your HawkEye AP system resides in the headquarters zone. Set up a daisy chain so that the Boston Collector can provide event-log data to the headquarters Collector for loading into the EDW. In this example of a daisy-chain configuration, the Boston Collector is the *source*, and the headquarters Collector is the *target*.



To configure the source Collector in the remote network zone:

1. While logged in as the `hexis` user, open `config.xml` in your editor.

For example:

```
vi <HawkEye AP Home>/etc/collector/config.xml
```

2. Define a *daisy-chain* log queue. For information, see “Configuring Log Queues”, on page 25.

3. Define retrievers that gather data from log hosts in this remote network zone, and specify the `daisychain` log queue within their `<LogQueue>` elements.

For more information, “Defining Retrievers”, on page 27.

4. Define a *daisy-chain* loader.

Daisy-chain loaders create tar files with a log file and corresponding and meta file. They place the tar files in the directory specified by the `<DaisyChainDir>` element. A daisy-chain retriever in the target Collector of the other zone picks up the special tar files from this staging directory.

For example:

```
<Loader name="daisychain" enabled="1">
   <LogQueue>daisychain</LogQueue>
   <DaisyChainDir>/tmp/daisychain</DaisyChainDir>
</Loader>
```

To configure the target Collector in the local network zone:

1. While logged in as the `hexis` user, open `config.xml` in your editor.

For example:

```
vi <HawkEye AP Home>/etc/collector/config.xml
```

2. Define a *daisy-chain* log queue. For more information, see “Configuring Log Queues”, on page 25.

3. Define the *daisy-chain* retriever of type SFTP.

Daisy-chain retrievers collect special tar files and use a provided preprocessor to convert the tar file back into the `.log` and `.meta` file pair, which retrievers write to the specified log queues. The `<SourceHost>` and `<SourceUser>` elements allow the daisy-chain retriever to connect to the remote Collector host as a specified user. The remote host must allow SFTP for the provided user without passing a password. The `<keepAlive>` attribute keeps the connection open to check for new files.

IMPORTANT: The Preprocess ID in the following example is set to 1. Daisy-chain retrievers must take precedence over other retrievers.

```
<Retriever name="daisychain" type="sftp" enabled="1"
   deleteOriginal="1"  keepAlive="1">
   <SourceDir>/tmp/daisychain</SourceDir>
   <SourceHost>host01.boston.myco.com</SourceHost>
   <SourceUser>hexis</SourceUser>
   <LogQueue>daisychain</LogQueue>
   <!-- Preprocessor to convert .tar files into .log and .meta files -->
   <Preprocess id="1" type="daisychain">DaisyChain</Preprocess>
   <!-- Daisy-chain Collector only collects .tar files; .wrk files are used
      in the process of .tar file creation -->
   <Plugin name="Default">
      <File id="1" ai="ignore">.*</File>
      <File id="2" ai="accept">tar$</File>
   </Plugin>
</Retriever>
```

TIP: Setting the `deleteOriginal` attribute to `1` (true) enables the Collector to maintain the daisy-chain directory and minimize disk-space usage by only storing log files that have not been processed.

4. Define a file system loader to load the files collected by the daisy-chain retriever.

For example:

```
<Loaders>
  <Loader name="fs" enabled="1">
    <LogQueue>daisychain</LogQueue>
   <PTL table="web" type="file" enabled="1">/usr/apache_access-1BT.ptl</PTL>
    <SLSInstance>slshost.myco.com:2112</SLSInstance>
  </Loader>
```

```
        </Loaders>
```

CONFIGURING ERROR REPORTING

You can configure error reporting for Activity and Error logs in `<`*`HawkEye AP Home`*`>/etc/collector/config.xml`. Here, you can specify:

- The location of activity and error logs as noted below.

Specifying the location of activity.log and error.log

By default, these logs live in `<`*`HawkEye AP Home`*`>/var/log/collector/`. To change that location, edit the path that appears in the following lines of the `config.xml` file:

```
<File>
    <Activity><HawkEye AP Home>/var/log/collector/activity.log</Activity>
    <Error><HawkEye AP Home>/var/log/collector/error.log</Error>
</File>
```

For more information on setting up Collector logging, see see “About Collector logs”, on page 130 of the *Administration Guide.*

ADJUSTING PERFORMANCE LEVELS

The following elements defined in the `config.xml` file set the interval between Collector cycles and flushing files to disk, as well as maximums in terms of system use and processes. You should only have to alter these settings when responding to a performance issue. Contact Technical Support for guidance.

CycleDelay

The number of seconds to delay between cycles of the internal Collector. Controls how often the Collector checks for changes to Retriever and Loader status, and sends alert emails. Defaults to 15 seconds if no value is specified. If you need to increase the value to improve performance, the value should be less then the interval specified for Loaders and Retrievers using the `PollInterval`, `Period`, `Schedule` elements or the `UnchangedFor` attribute (“Scheduling and the Unchanged For Attribute”, on page 138 of the *Administration Guide*.

```
<CycleDelay>15</CycleDelay>
```

For information about the Dashboard file, see Monitoring Logs in Chapter 6: Administering the Collector of the *Administration Guide*.

Throttle

To throttle the system load level and system processes, add or edit the following line in `config.xml`:

```
<Throttle load="5" process="500"/>
```

In the definition above, the `load` attribute specifies the maximum load that your CPU can handle. The `process` attribute is the maximum number of processes that the available memory on your system can handle. The number of processes refers to *all* the processes running on your system, not just those processes that belong to the Collector.

When adjusting throttle attributes, Hexis Cyber Solutions recommends that you use the highest possible setting. Then work down to the level that best maximizes performance for your system.

TIP: Paying attention to your network bandwidth usage is a better way to improve Collector performance issues. In general, you shouldn't need to adjust the Throttle setting.

FlushInterval

The flush interval sets the number of cycles between flushes to disk of internal state. Flushing more often lowers likelihood of loss of state in the event of hardware failure, but may require more CPU and I/O. To disable flushing altogether, set the value to `-1`. To turn on automatic flushing, which flushes to disk after every change in state, set the value to `0`.

```
<FlushInterval>10</FlushInterval>
```

CHAPTER 3

Configuring Retrievers and Receivers

This chapter contains these sections:

- "Configuring HawkEye Retriever on Windows for Batch Processing", next
- "Using the NetFlow Receiver", on page 59

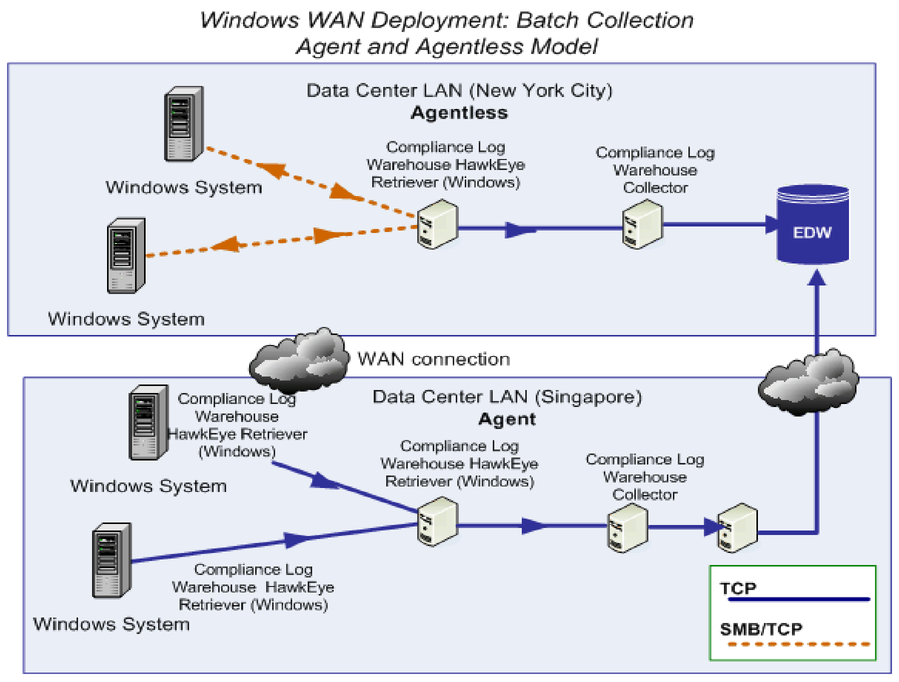
Configuring HawkEye Retriever on Windows for Batch Processing

The HawkEye Retriever on windows can be configured in two ways for batch processing as shown in Figure 2-11:

- **agentless**—polls streams of event-log data from Windows systems
- **as an agent**—pushes streams of event-log data and application data from Windows systems

In spite of its name, the HawkEye Retriever functions as a receiver as well as a retriever. It enables your site to pass events from Windows to the HawkEye Collector for batch processing.

Figure 3-1


Windows WAN Deployment: Batch Collection
Agent and Agentless Model
Data Center LAN (New York City)
Agentless
Windows System
Compliance Log Warehouse HawkEye Retriever (Windows)
Compliance Log Warehouse Collector
EDW
Windows System
WAN connection
Data Center LAN (Singapore)
Agent
Compliance Log Warehouse HawkEye Retriever (Windows)
Windows System
Compliance Log Warehouse HawkEye Retriever (Windows)
Compliance Log Warehouse Collector
Compliance Log Warehouse HawkEye Retriever (Windows)
Windows System
TCP
SMB/TCP

You cannot forward data directly from the Retriever agent on a Windows host to the HawkEye Collector. You must forward the data from the agent to a HawkEye Retriever that runs on a Linux machine. In this case, you are chaining one HawkEye Retriever to another. For information on configuring this chaining, see "Defining a TCP Event Writer", on page 95.

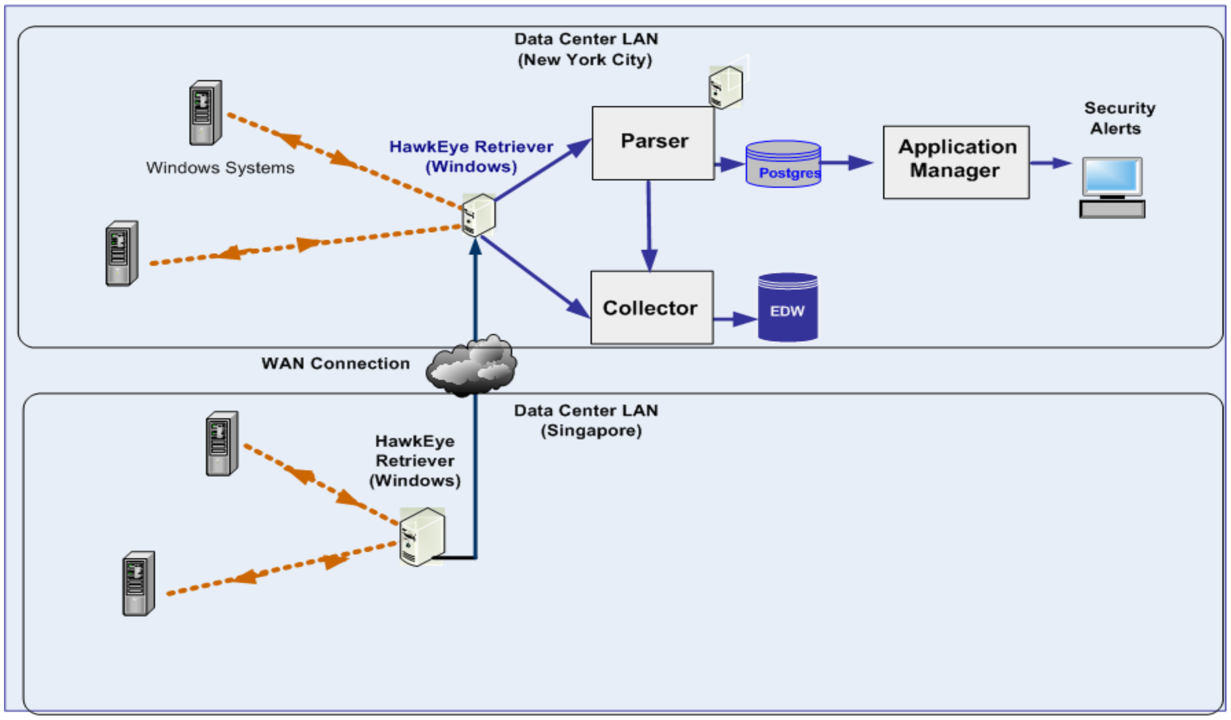
In a Windows LAN Configurattion, you can configure the HawkEye Retriever to pass the collected data to a log queue for batch processing by the Collector or information on batch processing of Windows event-log data, see "HawkEye Retriever on Windows", on page 34.

Windows Agent and Agentless Configuration

Figure 3-2 illustrates Windows servers distributed between two data centers. However, Figure 3-2 illustrates agentless configuration in one data center and agent configuration in the other. When you configure the retriever as an agent, you can collect Windows application data as well as event-log data from the Windows host.

When you configure the HawkEye Retriever as an agent on a Windows system, you must configure it to forward its data to a HawkEye Retriever that runs on a Linux host. Figure 3-2 illustrates a retriever configuration that defines two HawkEye Retrievers: one as an agent on Windows and the other agentless on Linux.

Figure 3-2: Windows WAN Deployment—Streaming Collection for Agent and Agentless Model



For more information on configuring the retriever as an agent, see:

- agent configuration process -- "Configuring the Retriever to Run as an Agent", on page 97.
- chaining configuration -- "Defining a TCP Event Writer", on page 102

USING THE NETFLOW RECEIVER

The Netflow Receiver gives Network Managers a detailed view of Application Flow on the Network. For a general description of the Netflow Receiver, see “Collecting and Analyzing Network Data with the Netflow Receiver”, on page 18. For details on configuring Netflow Receiver, see “Configuring Netflow Receiver”, on page 124.

CHAPTER 4

HawkEye Retriever Configuration

This chapter contains the following sections:

NOTE: Setting up HawkEye Retriever requires you to set up the configurations described in this document as well as those in the Log Adapter setup instructions (such as the microsoft_windows_securityEvent_sensageRetriever Log Adapter). As an example, see Setting up the microsoft_windows_securityEvent_sensageRetriever Log Adapter in Chapter 22, “microsoft_windows_securityEvent_sensageRetriever,” in the *Analytics Guide*.

STANDARD CONFIGURATION

This section describes the following topics:

Introduction

The HawkEye Retriever is used to collect data from a variety of sources and then write that data to one or more destinations. The HawkEye Retriever can either run centrally on a Collector that is pulling data from remote hosts, or it can be deployed as an agent on the remote host collecting data locally and then forwarding that data back to HawkEye AP

Every instance of a HawkEye Retriever (running remotely as an agent on a target system or on a collector) must define the following:

- At least one event reader, which specifies a source of event data. See “Valid Source Data Types for Event Readers”, on page 62 and “Defining an Event Reader”, on page 66.
- At least one event writer, which specifies a destination for event data. See “Valid Destination Types for Event Writers”, on page 62 and “Defining an Event Writer”, on page 75.
- Pipeline settings that are specific to your data; a pipeline specifies the connection between each reader and its writer(s) and can be optimized for specific reader types and the unique pattern types of your environment. See “Using Pipelines”, on page 62 and “Defining a Pipeline”, on page 76.

Valid Source Data Types for Event Readers

You can define a reader to accept several types of source data:

- Windows Event Data (over SMB) connection
- SQL Server audit data
- SQL Server 2000 audit data (as a Remote Agent only)
- Generic SQL Server table data
- Oracle Audit Log (BETA)
- Flat files matching a search pattern or regular expression
- Data sent over a TCP socket. Used most often to daisy chain this HawkEye Retriever to another HawkEye Retriever

The reader has a different set of properties for the different source types. For more information, see "Defining an Event Reader", on page 66.

Valid Destination Types for Event Writers

When you run the HawkEye Retriever on a HawkEye Collector, you can define an event writer to send the data to a Collector log queue for batch collection.

When you run the HawkEye Retriever on Windows, you can define an event writer to send the data to one of three destination types:

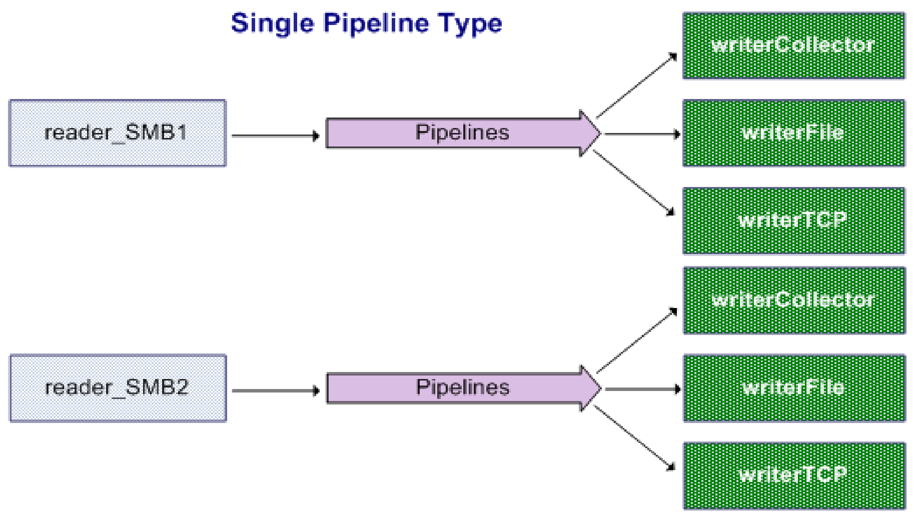
- Flat file
- syslog writer
- TCP socket for chaining to another HawkEye Retriever

The event writer has a different set of properties for each destination type. For more information, see "Defining an Event Writer", on page 75.

Using Pipelines

When you use the standard process to configure HawkEye Retriever with a single pipeline type, a pipeline is automatically created between each event reader and every event writer that you define. In other words, there is a unique pipeline for each reader and the data from every defined data source is sent to all of your defined destinations. Figure 4-1 illustrates this configuration.

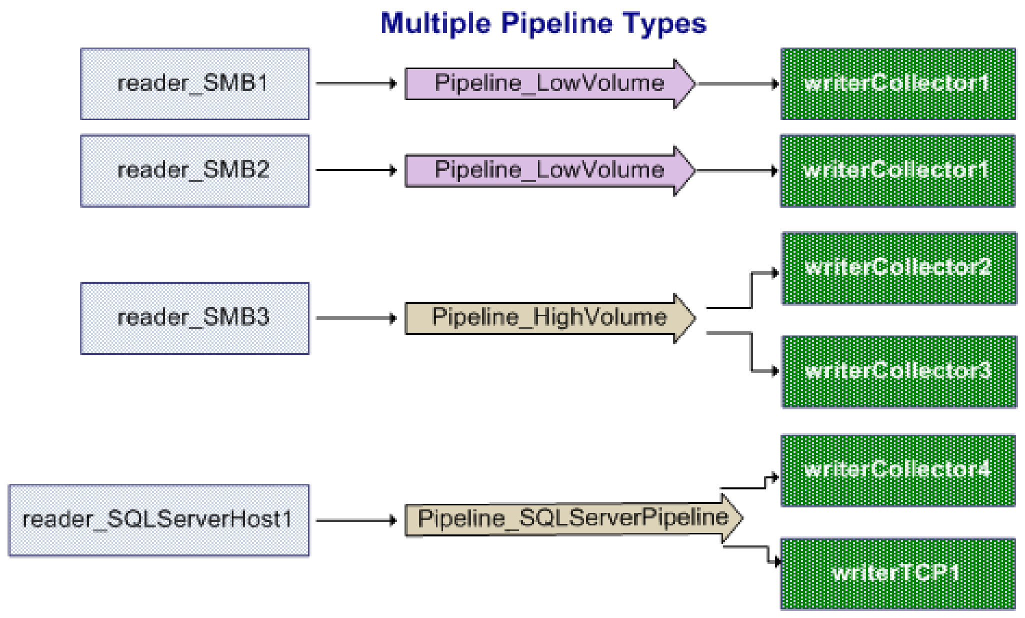
Figure 4-1: Example Configuration with Single Pipeline Type



Although standard configuration creates a separate pipeline for each reader you define, each pipeline uses the same configuration values. In some cases, this single set of pipeline values will not meet your collection needs. Your site may need specific pipeline values for specific readers. For example, assume some of your SMB readers have significantly higher traffic than others. In this case, you will want to configure two different pipeline types, one for the high-volume and one for low-volume readers. Another case in which a single pipeline type does not suffice, is when you have a mix of different reader types such as Windows Event readers and SQL Service Audit Log readers. Then you will want to configure two different pipelines to accommodate each of these readers.

You can use the standard configuration process to create different types of pipelines. In this case, you must assign a specific pipeline type to each reader and connect that reader to one or more writers. Figure 4-2 illustrates this configuration.

Figure 4-2: Example Configuration with Multiple Pipeline Types



For standard configuration of a single pipeline type, there are only three pipeline properties to specify and all three properties have a default value. Standard configuration of multiple pipeline types uses the same three properties but also requires you to name each pipeline type and identify its writer(s).

The standard process enables you to configure and run the HawkEye Retriever easily and quickly. Hexis Cyber Solutions recommends, however, that you analyze your data against the default pipeline values to determine the best settings for your site. For more information, see the following:

- **Standard Configuration**—“Defining a Pipeline”, on page 76
- **Advanced Configuration**—“Pipelines”, on page 97

Retriever Definition Files

To configure the HawkEye Retriever you must define its event readers and event writer by setting the appropriate properties for its type. You define these properties in a plain text file. HawkEye AP provides two example files:

- **agentless configuration**—`agentless.prop`

  The most common configuration is to have the Collector manage the HawkEye Retriever, regardless of whether you define the HawkEye Retriever for batch or real-time processing. In this case, you run the retriever agentless on a Linux host.

- **agent configuration**—`agent.prop`

  Alternately, you can configure the HawkEye Retriever to run as an agent on a Windows system.

In more complex environments you may have several prop files defined. Each prop file would be referenced by the Collector config file in a different HawkEye Retriever element, and each would define a unique HawkEye Retriever instance.

The following sections describe both of these configurations.

Attributes for HawkEye Retrievers

This section documents the required and optional sections of the `agentless.prop` and `agent.prop` property files and provides examples:

- "Introduction", next
- "Defining an Event Reader", on page 66
- "Defining an Event Writer", on page 75
- "Defining a Pipeline", on page 76
- "Defining Remote Shell Access", on page 85
- "Defining the Logging Level", on page 85
- "Defining the State Files", on page 86
- "Controlling Historical Data", on page 86

Introduction

The `agentless.prop` and `agent.prop` property files facilitate the most common configurations, which consist of one or more event readers, pipeline types, and event writers. These files simplify configuration by providing default values for most properties. As mentioned above in "Using Pipelines", on page 62, when you define multiple pipeline types, you must name each pipeline type and identify its writers.

The default values are often not explicit in these standard settings. For example, if you define a flat-file event writer, you specify only its type (`file`) and its path and name. By default, the HawkEye Retriever appends log data to the file you specify. The standard syntax is:

```
writer=file,<file_name>
```

To change the default setting, you must specify the full set of advanced properties that define this type of event writer. Advanced properties for the HawkEye Retriever and its submodules are documented in "Advanced Configuration", on page 88.

The example below illustrates the full syntax required to define a flat-file event writer that *overwrites* the output file.

```
# Declare the writer
event.writers=writerFile
# Define the fileWriter
event.writer.writerFile.class=com.sensage.eventlog.writer.FileEventWriter
event.writer.writerFile.filename=/<full_path>/output.log
event.writer.writerFile.append=false
event.writer.writerFile.serializer.class=com.sensage.eventlog.serializer.Delimit
edSerializer
```

NOTE: There are two ways to use the standard configuration:

- Define a single pipeline type—called "`pipelines`".

Use this configuration when all readers use the same pipeline values and all readers write to all writers.

- Define multiple pipeline types, each with its own values.

  Use this configuration when one set of readers uses one pipeline type and another set of readers uses a different pipeline type. In this case, all readers may not write events to all available writers. In other words, a writer can be used by multiple pipeline types but a reader always uses a single pipeline type.

WINDOWS AUTO-ARCHIVE

NOTE: The HawkEye Windows Retriever is NOT supported to work when Windows Event Viewer is configured to automatically archive the event log when full. Implementing Windows Event View saves the Windows Events into a separate file that cannot be retrieved automatically. Instead, configure the Windows Event Viewer to "Overwrite events as needed (oldest events first)" and configure a maximum log size sufficiently large that, when combined with the configured frequency of HawkEye Windows Retriever, there will be minimal if any chances that events are missed.

Windows 2008R2 and NTLMv2

Windows 2008R2 deprecated several security paradigms from earlier Windows releases. If your Windows2008R2 servers require the use of NTLMv2 authentication, you need to add a new flag to your prop file:

```
event.readers.usentlmssp=true
```

This flag can be added anywhere in the file and instructs the HawkEye Retriever to use the newer NTLMSSP security provider when making an NTLMv2 authentication to the Windows server. After you make the change, your NTLMv2 Windows hosts should begin to pull data correctly.

Defining an Event Reader

Every HawkEye Retriever must have at least one event reader. Each event reader specifies the source of event data. The table below documents the available reader types and their syntax.

Event Source	Syntax
Windows SMB connections	`reader<suffix>=windows,<host>,<username>,<password>[,<log_type>[:<log_type>[...]][,<pipeline_name>]` where `<log_type>` can be any of the following: • **`sec`** — Security, **`sys`** — System, **`app`** — Application, **`dns`** — DNS Server, **`frs`** — File Replication Service, **`dir`** — Directory Service • reader - the valid reader name in which **reader** is appended with a <*suffix*> that includes text as in **reader1=windows** or with text that includes optional underscores as in **reader_HOSTA=windows** • hostname - IP address or WINS name of the Windows host from which to connect using SMB • username - the user name for SMB authentication • password - the password for SMB authentication • pipeline_name - only used when different readers require different pipeline types. See“Defining a Pipeline”, on page 76. **NOTE**: • If you specify more than one log type, use a colon separator; for example, to cause the HawkEye Retriever to read only Security and Application events, specify `sec:app`. • You can specify multiple log types in any order. • If you do not specify log type(s), the retriever reads all log types.
Oracle Audit Log Data (BETA) **Note:** The Oracle Audit Log Reader can be used to read data from any Oracle table by using a named SQL string that references the tables that you want to query.	`reader<suffix>=oracleauditlog,<hostname>,<username>,<password>,<servicename>,<port>,<sql string name>,[<event type>],<password manager>,<daysprevious>,[,<timestamp column name>][,<pipeline_name>]` • reader - the valid reader name in which **reader** is appended with a <*suffix*> that includes text as in **reader1=oracleauditlog** or with text that includes optional underscores as in **reader_HOSTA=oracleauditlog** • hostname - the IP address or FQDN of the Oracle host • username - The Oracle account used to access the table (must have Select permissions to that table) • password - the password supplied for this user (after the collector starts this string is replaced with a set of ***'s** and the password is stored in a separate keystore file) • servicename - the Oracle service name you wish to collect data from • port - the port used to contact the Oracle host • sql string name - references a string in the sqlstringmap to use to query the server. • event type - see the section “Setting an Event Type”, on page 74 for more information. • password manager - if not left blank, this value should be the name of a defined password manager that will be used for this reader. • daysprevious - a number (0 or higher) that indicates how many days in the past to start pulling data the first time this reader is used. 0 would indicate that you want to only pull new data, a 5 would indicate you want data as far back as 5 days ago • timestamp column name - this optional flag specifies the column name of a column that has a timestamp the retriever can use to track data it has already read. The default is the extended_timestamp field of the oracle audit logs • pipeline_name - only used when different readers require different pipeline types. See“Defining a Pipeline”, on page 76.

Event Source	Syntax

SQL Server Generic Table Data	`reader<suffix>=sqlservertable,<hostname>,<username>,<password>,<port>,<dbname>, <sql string name>,<ts column>,[<event type>],[,<pipeline_name>]`
	
	• reader - the valid reader name in which **reader** is appended with a <*suffix*> that includes text as in **reader1=sqlservertable** or with text that includes optional underscores as in **reader_HOSTA=sqlservertable**
	• hostname - the IP address or FQDN of the SQL Server host
	• username - The SQL Server account used to access trace files (must have READ permissions on trace files)
	• password - the password supplied for this user (after the collector starts this string is replaced with a set of ***'s** and the password is stored in a separate keystore file
	• port - the port used to contact the SQL Server host
	• databasename - A database on SQL Server.
	• sql string name - references a string in the sqlstringmap to use to query the server.
	• trx column - Specifies the name of a column that is of type extended timestamp that the EDW will use to query and order the data. To define an alias column as <*tx column*> wrap the alias expression around {{}}. For example:
	•
	`SELECT [id], [desc], {{convert(datetime,([startdate] + ' ' + [statetime]))}} as start_date FROM [dbo].[Test] where [EXTENDEDTSCOL]`
	Note: start_date can be used for the <*ts column*>.
	• event type - see the section "Setting an Event Type", on page 74 for more information.
	• pipeline_name - pipeline_name - only used when different readers require different pipeline types. See"Defining a Pipeline", on page 76.
	
	Note: By default logs collected from a generic SQL Server table and sent to a collector type writer will have the log prefix of 'unknown', for these readers it would be useful to create a custom log type so that the files created are prefixed with something relevant to the data being queried.

Event Source	Syntax

SQL Server Audit Traces	`reader<suffix>=sqlserverauditlog,<hostname>,<username>,<password>,<port>,{<metaDataFunctionName>],[<max records per poll>],[<event type>],[<tscolumn>],<trace definition file>,[,<pipeline_name>]`
	
	• reader - the valid reader name in which **reader** is appended with a <*suffix*> that includes text as in **reader1=sqlserverauditlog** or with text that includes optional underscores as in **reader_HOSTA=sqlserverauditlog**
	• hostname - the IP address or FQDN of the SQL Server host
	• username - The SQL Server account used to access trace files (must have READ permissions on trace files)
	• password - the password supplied for this user (after the collector starts this string is replaced with a set of ***'s** and the password is stored in a separate keystore file
	• port - the port used to contact the SQL Server host
	• metaDataFunctionName - the name of a function on the server that returns meta data about the **.trc** files. It must exist on the SQL Server host. If it does not exist, then the trace is read according to the default, which can have performance implications. See Note 1 below for more details. Also refer to the following section "Defining a Metadata Function", on page 117, for performance-related details.
	• max records per poll - the maximum number of records that can be brought back in a single polling interval. Defaults to 25000. See Note 2 below for more details.
	• event type - see the section "Setting an Event Type", on page 74 for more information.
	• ts column - an optional column name for the column in the trace that has a valid timestamp. Normally not set.
	• trace definition file - last entry unless a pipeline name is added. This is a file name that points to a file containing a trace definiton. See Note 3 below or details.
	• pipeline _name - only used when different readers require different pipeline types. See"Defining a Pipeline", on page 76.
	
	NOTE1:
	• The metaDataFunctionName must have a signature, as In:
	`     ALTER FUNCTION [dbo].[getTraceFiles](@folder`
	`nvarchar(260),@tracefilename nvarchar(260))`
	`     RETURNS @TraceFileMetaData TABLE`
	`(`
	`     [fileName] VARCHAR(260),`
	`     [size] BIGINT,`
	`     [LastModified] DATETIME,`
	`     [error] VARCHAR(500)`
	
	• When the metaDataFunctionName is called, it must return a list of file names, sizes, and last modified dates for all **.trc** files in the directory, specified in the "folder" having the base name specified in *tracefilename*.
	• Hexis Cyber Solutions provides a sample function for metaDataFunctionName that covers most customer use. Because there are alternative ways to modify this function that have security implications, keep in mind when choosing to modify the function that you choose a solution that best addresses security impacts. Refer to the following section "Defining a Metadata Function", on page 117, for performance-related details.

Event Source	Syntax

	NOTE 2:
	• When setting max records per poll the goal is efficiency in querying the **.trc** files. If you are polling less often and trace files are large (that is 100MB or larger) you may want to increase the default setting of 25000. For details on tuning non-SMB readers see "Tuning Non-SMB Readers", on page 84.
	• Increasing the max records per poll setting may require that you provide more memory to the Java application; however, the trade off is less load on the SQL Server.
	• If the max records per poll is too large, you can easily use up all available memory on the retriever instance. This probably means you have the value set too high, but you can increase the amount of memory available to SQL server by modifying the `<sensage_prefix>/bin/windowsretriever` file.
	
	A call that invokes the WindowRetriever jar is in the last line as noted below:
	
	`"$SENSAGE_JAVA_HOME/bin/java"\`
	`-cp ${CLASSPATH}\`
	`"$@"\`
	`com.sensage.eventlog.retriever.EventRetriever`
	
	Insert a line so that it looks like this:
	
	`"$SENSAGE_JAVA_HOME/bin/java"\`
	`-Xms1024m -Xmx1024m\`
	`-cp ${CLASSPATH}\`
	`"$@"\`
	`com.sensage.eventlog.retriever.EventRetriever`
	
	`See the Java documentation for more details on the affects and concerns when using the Xms and Xmx directives.`
	
	NOTE 3:
	• Regardless of how the trace file location is created, the name of the trace file is specified once using this tag:
	`DECLARE @TRACE_BASEFILENAME NVARCHAR (245)`
	`SET@TRACE_BASEFILENAME = ’nameofthetrace’`
	• The trace name can be assigned to different drive letters or paths, as long as the **SET @TRACE=** only occurs once, as in:
	`DECLARE @TRACE_BASEFILENAME = ’sqlauditfile’`
	`IF <some condition>`
	`SET @traceFileLocation = ’c:\trace\’ + @TRACE_BASEFILENAME`
	`ELSE`
	`SET @traceFileLocation = ’u:\trace\’ + @TRACE_BASEFILENAME`
	
	• A SQL Server trace definition is a collection of *n* number of files, each *n* megabytes in size.
	• Based on the file meta data from the .metaDataFunctionName function, the size of each **.trc** file (obtained from SQL Server),the last timestamp that was read, and the max records per poll, the retriever will create a query that spans a minimum number of files to complete the query.
	• For example, if the max records per poll is 25,000 and each log file is 100MB, then the query will be restricted to at most two of the **.trc** files.
	• Smaller values from the max records mean faster queries, but for a busy server they will also mean less efficient use of the system since the server will need to pull all data from the two **.trc** files into memory during each poll, even if only a fraction of the records are retrieved.

Event Source	Syntax
SQL Server 2000 Audit Traces	`reader<suffix>=sqlserver2000auditlog,<hostname>,<username>,<password>,<port>,[<event_type>],<trace definition file>,[,<pipeline_name>]` • reader - the valid reader name in which **reader** is appended with a <*suffix*> that includes text as in **reader1=sqlserver2000auditlog** or with text that includes optional underscores as in **reader_HOSTA=sqlserver2000auditlog** • hostname - the IP address or FQDN of the SQL Server host • username - The SQL Server account used to access trace files (must have READ permissions on trace files) • password - the password supplied for this user (after the collector starts this string is replaced with a set of ***'s** and the password is stored in a separate keystore file • port - the port used to contact the SQL Server host • trace file - path to a file that defines the sql server trace • ts column - Specfies the name of a column that is of type extended timestamp that the EDW will use to query and order the data. • event type - see the section “Setting an Event Type”, on page 74 for more information. • pipeline _name- only used when different readers require different pipeline types. See“Defining a Pipeline”, on page 76.
Flat files	`reader<suffix>=file,<file_name>,[<event type>] [,<pipeline_name>]` • reader - the valid reader name in which **reader** is appended with a <*suffix*> that includes text as in **reader1=file** or with text that includes optional underscores as in **reader_HOSTA=file** • File_name - Can be any of the following: • an OS-appropriate path to a single file name (for example: **C:\temp\log.out)** • A string that represents a set of possible paths and/or directories using a **?** (question mark) to indicate a single character and a ***** (asterisk) to indicate multiple characters (for example: **/tmp/dir?/logifle*.log**) • A valid expression that specifies a group of files and/or directories to scan (for example: **/var/log/myapp/day??/[Ll]ogF(l|ile).*\.log)** • event type - see the section “Setting an Event Type”, on page 74 for more information. • pipeline _name- only used when different readers require different pipeline types. See“Defining a Pipeline”, on page 76. **NOTE**: 1. Path names should be specified using the appropriate path separator for that operating system. For example on windows you should specify a drive letter and use the backslash, **'c:\temp\file',** and on Linux you would use the forward slash, **'/tmp/logfile.out'** 2. For more advanced features you must use the file reader syntax outlined in the “Advanced Configuration”, on page 88.

Event Source	Syntax
Multiple flat files	`reader<suffix>=multifile,<search_string>,[<event_type>[,<pipeline_name>]]` • reader - the valid reader name in which **reader** is appended with a <*suffix*> that includes text as in **reader1=multifile** or with text that includes optional underscores as in **reader_HOSTA=multifile** • search_string - a string that represents a set of possible paths and/or directories. By default, the string assumes every character is literal, except you can use a **?** (question mark) to indicate a single character and a ***** (asterisk) to indicate multiple characters (for example: **/tmp/dir?/logifle*.log**) • event type - see the section “Setting an Event Type”, on page 74 for more information. • pipeline _name- only used when different readers require different pipeline types. See“Defining a Pipeline”, on page 76. **NOTE**: 1. Path names should be specified using the appropriate path separator for that operating system. For example on windows you should specify a drive letter and use the backslash, **'c:\temp\file',** and on Linux you would use the forward slash, **'/tmp/logfile.out'** 2. You can optionally use a regex to define the search. Refer to the “Advanced Configuration”, on page 88 for details. 3. The MultiFile reader will scan, in parallel, files from multiple matching directories. For more advanced features you must use the multifile reader syntax outlined in the “Advanced Configuration”, on page 88.
TCP sockets	`reader<suffix>=tcp,<port>[,<pipeline_name>]` • reader - the valid reader name in which **reader** is appended with a <*suffix*> that ncludes text as in **reader1=tcp** or with text that includes optional underscores as in **reader_HOSTA=tcp** • port - the contact port. for the TCP socket. • pipeline _name- only used when different readers require different pipeline types. See“Defining a Pipeline”, on page 76.

IMPORTANT:

- Specify a <*pipeline_name*> only when different readers require different pipeline types. When you specify different pipeline types, you must delete the line in the property configuration file that defines the default pipelines. For more information, see “Defining a Pipeline”, on page 76.
- You can set a flag that causes all new SMB readers to pull over only new records and not large volumes of historical data. This flag can also help prevent pulling duplicate data in the event you migrate readers across collectors or have lost the state database.

  ```
  event.reader.startallreadersfromlatest=true;
  ```

- All three sqlserver related readers gather all data on the devices unless the **startallreadersfromlatest** or **startnewreadersfromlatest** properties are set. This means:
  - With neither property set: all data is pulled, which pulls older data.
  - With **startnewreaderfromlatest** set to true: All data is pulled starting from "now" for a new reader.This is the most common scenario.
  - With **startallreadersfromlatest**: All data is pulled starting from "now" every time the collector is accessed. This scenario is best when you need to guarantee that only new data is pulled.

- For standard configuration, all reader names must be unique and the first letters of a reader name must be "reader"; in other words, the name **readerSMB** is valid but **SMBreader** and **ReaderSMB** are invalid. Other examples of valid reader names are: **reader_smbhost1**, r**eaderMyHostA** and **reader1**.
- **WARNING:** As a best practice avoid routinely reusing reader names. If you do reuse them the retriever's state information may contain a previously named reader, which causes confusion in interpreting the actual name. Note that defining a reader with the name **reader1=** is perfectly valid, but only if this name has not been previously used.

Examples of each type of event reader:

- Windows SMB connections

  ```
  readerSMB=windows,10.0.1.15,bob,bobpassword
  ```

- Oracle Audit Logs (BETA)

  ```
  readerOracle=oracleauditlog,10.0.1.15,bob,bobpassword,xe,1521,dbacommonsql,ora
  cleevents,,4,oracle_pipeline
  sqlstringmap.dbacommonsql=select STATEMENT_TYPE,DB_USER,EXTENDED_TIMESTAMP,
  COMMENT_TEXT from dba_common_audit_trail where [EXTENDEDTSCOL]
  ```

- SQL Server Generic Table Data

  ```
  readersqltbla=sqlservertable,10.0.1.15,bob,bobpassword,1433,sqlserverdatabase,
  tableAsql,mytscolumn,tableA,sqlserver_pipeline
  sqlstringmap.tableAsql=select * from tableA where [EXTENDEDTSCOL] and userid =
  'sally'
  ```

- SQL Server Audit Trace

  ```
  readerSQLServerhostA=sqlserverauditlog,10.0.1.15,bob,bobpassword,1433,getTrace
  Files,30000,sqlserver_events,,/tmp/mytrace.sql,pipeline_sqlserver
  ```

- SQL Server 2000 Audit Trace (this must be run as a remote agent)

  ```
  readerSQLServer2KhostB=sqlserver2000auditlog,10.0.1.15,bob,bobpassword,1433,25
  000,sqlserver2000,,/tmp/mytrace.sql,pipeline_sqlserver
  ```

- Flat files
  - Single pipeline-type configuration:

    ```
    readerFile=file,/<full_path>/sample.log
    readerFile=file,/var/log/sample*.log
    ```

  - Multiple pipeline-type configuration:

    ```
    readerFileLow=file,/<full_path>/sample.log,pipeline_lowvolume
    readerFileHigh=file,/<full_path>/sample.log,pipeline_highvolume
    readerFileLow=file,/var/log/sample*.log.pipeline_lowvolume
    readerFileHigh=file,/var/log/sample*.log.pipeline_highvolume
    ```

- TCP sockets

  ```
  readerTCP=tcp,9999
  ```

IMPORTANT: Both property files provide examples and a commented line for one reader. You must remove the # symbol to enable the line and then provide a value for this property.

NOTE: If you have multiple readers of the same type, each must be uniquely named. For example:

```
readerSMB_SecSys=windows,10.0.1.15,bob,bobpassword,sec:sys:app
readerSMB_All=windows,10.0.1.16,alice,alicepassword
```

Setting an Event Type

An Event Type is a string (no special characters, that is, 0-9a-zA-Z) that helps identify a source of data. Each writer will then use or ignore the Event Type as appropriate. There are five built-in types for Windows data:

- sec (for Security)
- app (for Application)
- sys (for System)
- dns (for DNS Server)
- frs (for File Replication Service)
- dir (for Directory Service)

Readers allow you to also define a custom Event Type; for example, SQL Server Audit Log Readers (defined as **sqlserverauditlog**) could use the Event Type **sqlserver** and a File Reader that is reading Microsoft IIS logs might use the Event Type **iis**.

Note that different writers use Event Type differently:

- Collector Writers use the Event Type to name output files (which allows the Collector to determine which PTLS to apply to which files).
- TCP and Syslog Writers optionally prepend the Event Type to each record, so that downstream readers of the data can use the Event Type to determine the source.

Defining an Event Writer

Every HawkEye Retriever must have at least one event writer. Each event writer specifies the destination of event data that the readers supply. The table below documents the available event-writer types and their syntax.

Event Destination	Syntax
Collector log queue for batch collection	`<writer>=collector,<rolloverThresholdMinutes>,<rolloverThresholdBytes>[,<output_prefix>]` where • *<rolloverThresholdMinutes>* specifies the length of time that the retriever waits before giving the data to the Collector • *<rolloverThresholdBytes>* specifies the size of the log file before the retriever gives the data to the Collector HawkEye Retriever turns the data over to the Collector whenever the first of these values is reached. For example, if *<rolloverThresholdMinutes>* is set to `5` and *<rolloverThresholdBytes>* is set to `50000000`, the retriever watches the file until either it is 5 minutes old or 50,000 Mb in size. At this point, the retriever renames the file with a `.out` extension and is done processing the file. **NOTE**: Declare an *<output_prefix>* to prepend the specified string to the name of each output file that HawkEye Retriever creates. This option is useful to differentiate files that the Collector should process differently. For example, specify `win2k` as an output prefix for a writer assigned to a group of readers whose host is Windows 2000/2003, and specify `win2k8` for another writer assigned to a group of readers whose host is Windows 2008.
TCP socket for chaining to another HawkEye Retriever or for sending event to the real time system	`<writer>=tcp,<host>,<port>`
Flat file	`<writer>=file,<file_name>` **NOTE**: You must specify the full path to the file.
Syslog Writer	`<writer>=syslog,<host>,<port>`

IMPORTANT: For standard configuration, the first letters of a writer name must be **writer**. For example, the name **writerCollector** is valid but **CollectorWriter** and **WriterCollector** are invalid. Note that if more than one writer is declared, each *<writer-name>* must be unique and must begin with the word **writer**.

Examples of each type of event writer:

- Collector log queue
  - Single pipeline-type configuration—Specifies 60 minutes and 50,000,000-byte file size limit:

    ```
    writerCollector=collector,60,50000000
    ```

  - Multiple pipeline-type configuration—Specifies one writer for a group of Windows 2000/2003 hosts and another writer for a group of Windows 2008 hosts:

    ```
    writerCollector_win2k=collector,60,50000000
    writerCollector_win2k8=collector,60,50000000,win2k8
    ```

NOTE: Setting the optional prefix string enables HawkEye AP to use different PTLs when processing the different output files.

- TCP socket

  ```
  writerTCP=tcp,127.0.0.1,8099
  ```

- Flat file

  ```
  writerFile=file,/<full_path>/output.log
  ```

- Syslog Writer

  ```
  writerSyslog=syslog,127.0.0.1,514
  ```

IMPORTANT: Both property files provide examples and a commented line for the event writer. You must remove the # symbol to enable the line and then provide a value for this property.

Defining a Pipeline

Event pipelines specify the connection between event readers and event writers. As described above in "Using Pipelines", on page 62, you can use the standard configuration to define a single pipeline type that is used by all readers, or multiple pipeline types with different values that are assigned to specific reader(s).

DEFINING A SINGLE PIPELINE TYPE

When you use the standard process to configure the HawkEye Retriever for a single pipeline type, every pipeline has a unique reader and shares all defined writers.

For example, if you define one SMB reader that reads only Security and System events and another that reads all Windows log types, and you define two event writers (one that writes to a Collector log queue and the other to the Real Time system), each of your readers writes to both writers.

Because the actual source and destination is specified in the reader and writer respectively, the pipeline only needs to know the name of its reader and writer(s). The standard configuration process for a single pipeline type identifies these names automatically. When you define multiple pipeline types, however, you must specify all writers; and when you use the advanced configuration process, you must specify the reader and its writers.

The syntax for standard configuration of a single type of event pipeline is:

```
pipelines=<delay>,<requestcount>,<zerocounter>
```

All arguments are required. They define the following behaviors:

- **delay**—specified in milliseconds, represents the amount of time that the HawkEye Retriever waits before polling each reader for additional event records; use this argument to limit the load on the reader. Defaults to `1000`. For more information, see "Defining Delay and Request Count", on page 77.

- **requestcount**—specified as an integer, represents the maximum number of times the HawkEye Retriever polls each reader before the delay period begins; regardless of the number of records in the reader's buffer, each poll request returns a maximum of 100 event records. Defaults to `1`. For more information, see "Defining Delay and Request Count", on page 77.

  NOTE: The requestcount is required when defining a pipeline, but it only affects windows SMB readers, for non-SMB readers it has no effect.

- **zerocounter**—specified as an integer, represents the maximum number of times a reader can return zero (0) records before the HawkEye Retriever logs a warning in the log file indicating that the retriever might have fallen behind in gathering records. Defaults to `60`.

  NOTE: In the case of database readers (SQL Server), a warning is logged in the log file and an attempt is made to restart the trace as it is assumed the trace has stopped.

  For more information, see "Defining the Zero Counter", on page 79.

NOTE: Both property files provide the following defaults for this property:

```
pipelines=1000,1,60
```

Defining Multiple Pipeline Types

When you use the standard process to configure the HawkEye Retriever for multiple pipeline types, you must manually assign a specific pipeline to each reader. Additionally, because custom pipelines do not automatically send data to all writers, you must configure each pipeline type to identify each writer to which it sends the data.

For example, if you define three SMB readers that have high traffic and three that have low traffic, you will want to configure different pipeline types for the high-volume and low-volume readers.

In this case, because the different types of readers can write to a different subset of writers, you must specify the writer(s) in the configuration of each pipeline type. And because you are defining more than one pipeline, you must name each pipeline uniquely.

The syntax for standard configuration of multiple types of event pipelines is:

```
pipeline<suffix>=<delay>,<requestcount>,<zerocounter>,<writer>[,<writer>[...]]
```

All arguments are required. Three are identical to those specified for a single pipeline type. The arguments unique to a retriever that has multiple pipeline types are:

- **suffix**—uniquely identifies the pipeline, whose name must begin with "`pipeline`". The following are all valid: `pipeline1` and `pipeline2` or `pipeline_highvolume` and `pipeline_lowvolume`. There is no default for this value.

- **writer**—identifies every writer to which the pipeline sends data. There is no default for this value.

IMPORTANT: .Do *not* define values for both the `pipelines` option and named pipelines in the same property-configuration file or the Retriever logs an error and fails to start.

For examples of creating and using multiple pipeline types, see "Example of Creating and Using Multiple Pipeline Types", on page 80.

Defining Delay and Request Count

A pipeline is automatically created for each event reader; it uses the event writer as the common destination. You configure the HawkEye Retriever pipeline according to the number of events you receive regularly. You tune the retriever to poll for events only when needed so as not to waste network traffic or CPU processing but often enough to catch all events within a given time period.

IMPORTANT: If you define multiple readers and several of them have significantly higher traffic than the others, you should configure separate pipelines for the high-volume and low-volume readers. To configure multiple pipelines, you must use the advanced configuration process documented in “Advanced Configuration”, on page 88. In other words, when you use the standard configuration process, you cannot configure a pipeline separately for each reader, but once for all readers.

NOTE:
(1) If you choose to configure multiple pipeline types, read the conceptual material in this section first; then see “Pipelines”, on page 97, for information on creating multiple pipeline types. When you use advanced configuration to specify the pipeline types or any other Retriever submodule, you must use the advanced configuration process to specify attributes for all Retriever submodules.
(2). For file, TCP, and the database readers the only pipeline variable that matters is the polling interval. For more information on tuning the Database Retrievers see the section on “Using Database Retrievers”, on page 114.

When you configure the retriever pipeline, you specify how often the retriever should poll each reader for events before it initiates the delay. Because a reader cannot return more than 100 event records in a single poll request, you must ensure that you poll enough times to retrieve all events before the delay. However, you do not want to poll so frequently that the reader can do nothing but respond to poll requests.

By default, the pipeline waits 1000 milliseconds before it polls each reader for event records. For example, assume a reader receives approximately 100 events in 10 milliseconds. This frequency indicates that the retriever could receive more than 10000 event records in a second. Because the reader can return only 100 records in each poll request, you might configure the pipeline to poll the reader 100 times before the retriever pauses.

Assume further that the reader receives events inconsistently so that some seconds receive far fewer than 10000 records. You could extend the delay to `30000` milliseconds (30 seconds) to take advantage of the inconsistent flow. In other words, if you extend the delay so that the retriever stops polling for 30 seconds, you provide more time for it to average the records received over multiple seconds: those seconds with few records with those with the greatest number of records.

Extending the delay causes the retriever to pause longer between polls. Because it stops polling when a request returns zero (0) records, the retriever does not waste server time with unnecessary polls if the reader machine returns many records during each polling period. However, if the reader machine returns few records during each polling period, you should reduce the request count and increase the delay period.

There is a delicate balance between extending the delay period and increasing the request count:

- If your retriever is processing events from multiple readers and you are not using multiple processors, you must ensure that the delay is long enough to allow each reader sufficient time to read and return records. As a rule of thumb, assume you can set the delay period too short to allow all readers to process their records, but you cannot set it too long. It is important that all readers have idle time.

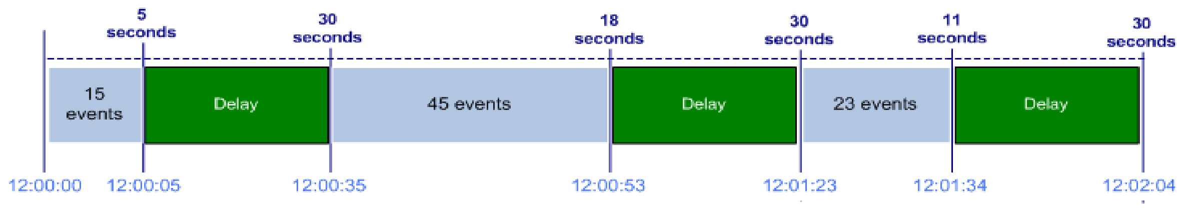
- If you increase the request count more than is necessary, you waste network and CPU time. As a rule of thumb, if a reader machine receives fewer than 100 records each minute, do not increase the default request count of `1`.

  For example, if you keep request count set to `1` and your readers typically return 30 or fewer records each polling period, the minimum record count ensures you receive all records in each poll. If you increase record count to `2`, the second poll will always return 0 records. In this second case, you have wasted machine and network time. You should poll the reader machines as infrequently as possible when they return 100 or fewer records in each polling period.

NOTE: As a general rule, do not reduce delay below `500` milliseconds. Instead, increase the request count.

The delay remains consistent regardless of the number of records received during each poll request. Figure 4-3 illustrates how delay behaves.

Figure 4-3: Illustrating a Delay of 30 Seconds



In the example above, the first poll returns 15 event records in 5 seconds, the second poll returns 45 event records in 18 seconds, and the third poll returns 23 event records in 11 seconds. Because the retriever waits 30 seconds after receiving data from the reader, the second poll begins 35 seconds after the first poll (5 + 30) at 12:00:35 and the third poll begins 18 seconds after the second poll (18 + 30) at 12:01:23.

IMPORTANT: You must test the value of these two arguments against your data to determine the best settings. The default values are only defaults and will probably not meet your needs. For more information, see:

- **testing your values**—"Testing Your Values", on page 82
- **setting delay and request count**— "Tuning SMB Event Retrieval", on page 83.

DEFINING THE ZERO COUNTER

On occasion, the retriever might receive no records over multiple consecutive poll requests. To ensure that the retriever does not erroneously miss data because of incorrect tuning, you must set a third pipeline argument: `<zero_counter>`. The value of this argument, specified as an integer, represents the maximum number of times a reader can return zero (`0`) records before the HawkEye Retriever logs a warning in the log file indicating that the retriever might have fallen behind in gathering records. The default is 60. For more information, see "Defining the Logging Level", on page 85.

NOTE: In the case of SQL Retriever, a warning is logged in the log file and an attempt is made to restart the trace on SQL Server as it is assumed the trace has stopped; be sure to keep the zerocounter value small so that the trace will start immediately in the event the trace has stopped. For example, assume you have set the zero counter to `10`. Assume also that you have four readers, as illustrated in the table below.

	reader1	reader2	reader3	reader4
poll request #1	10	10	10	0
poll request #2	10	3	0	0
poll request #3	10	5	3	0
poll request #*N*	...	...	...	0
poll request #10	10	8	5	0

As shown above, reader1 and reader2 return records in each poll request. Because reader3 returns no records in the second poll request, the retriever decrements the value of the zero counter for this reader to `9`. At the next poll, when reader3 returns three events, the retriever resets the value of its zero counter to `10`.

However reader4 never returns any records. At each poll, its zero counter decrements until it reaches zero. At this time, the retriever checks whether it has fallen behind in gathering event records. If it has, it logs a warning in the log file that is has fallen behind, skips ahead to the next available incoming event record, and triggers a system alert.

IMPORTANT: Because the traffic may differ widely among different reader machines, you may require separate pipelines for high-volume and low-volume machines to allow different settings for the zero counter. For example, if low-volume machines return fewer than 100 records each minute, they might return zero records in several consecutive polling periods. If you set the zero counter to the same value for these machines as for high-volume machines, you might erroneously trigger warnings in the log file.

Example of Creating and Using Multiple Pipeline Types

This section presents three different cases for multiple pipeline types.

Case 1: High and Low Traffic

This case assumes that your site has a collection of high volume servers (configured for readers 1-3) and lower volume servers (configured for readers 4-6) and that you want to use the standard configuration to poll the two groups of servers at different rates.

The configurations below illustrate how to configure the readers, the writer, and two pipelines, as well as three options.

```
#define your readers
readerSMB1=windows,10.0.0.1,Administrator,password,sec:sys,pipeline_highvol
readerSMB2=windows,10.0.0.2,Administrator,password,sec:sys,pipeline_highvol
readerSMB3=windows,10.0.0.3,Administrator,password,sec:sys,pipeline_highvol
readerSMB4=windows,10.0.0.4,Administrator,password,sec:sys,pipeline_lowvol
readerSMB5=windows,10.0.0.5,Administrator,password,sec:sys,pipeline_lowvol
readerSMB6=windows,10.0.0.6,Administrator,password,sec:sys,pipeline_lowvol

#define your writer
writerCollector=collector,50,500000

#define the two pipelines
pipeline_highvol=5000,10,60,CollectorWriter
pipeline_lowvol=30000,1,60,CollectorWriter

# remote access
remoteshell=9999,localhostOnly
#logging level
logging=INFO,../logs/winretriever.log
```

Case 2: Windows 2003 and Windows 2008 Servers

This case assumes that your site has a collection of Windows 2003 servers (configured for readers 1-2) and Windows 2008 servers (configured for readers 3-4). Each has roughly the same volume, but you want Windows 2008 events written to a file prefixed with the string '`win2k8`' so that it can be processed by a specific PTL.

The configurations below illustrate how to define the readers, the writer, and two pipelines, as well as two options.

```
#define your readers
readerSMB1=windows,10.0.0.1,Administrator,password,sec:sys,pipeline_win2k
readerSMB2=windows,10.0.0.2,Administrator,password,sec:sys,pipeline_win2k
```

```
readerSMB3=windows,10.0.0.3,Administrator,password,sec:sys,pipeline_win2008
readerSMB4=windows,10.0.0.4,Administrator,password,sec:sys,pipeline_win2008

#define two writers, one for 2008 and one for windows 2003/2000
writerCollector1=collector,50,500000
writerCollector2=collector,50,500000,win2k8

#define the two pipelines
pipeline_win2k=5000,10,60,writerCollector1
pipeline_win2008=30000,1,60,writerCollector2

# remote access
remoteshell=9999,localhostOnly
#logging level
logging=INFO,../logs/winretriever.log
```

Case 3: Windows 2003 and Windows 2008 Servers with Mixed Traffic

This case assumes that your site has the following servers:

- **Three Windows 2000/2003 with standard volume**—win2k_1, win2k_2, win2k_3

  NOTE: These servers send data to a Real Time Parse as well as to the Collector.

- **Three Windows 2000/2003 with low volume**—windesktop_1, windesktop_2, windesktop_3
- **Three Windows 2008**—win2k8_1, win2k8_2, win2k8_3

The excerpt below from the `agentless.prop` file provides an example of how to define and set the readers, writers, and pipelines for this environment.

```
#define your readers
readerSMB1=windows,10.0.0.1,Administrator,password,sec:sys,pipeline_win2k
readerSMB2=windows,10.0.0.2,Administrator,password,sec:sys,pipeline_win2k
readerSMB3=windows,10.0.0.3,Administrator,password,sec:sys,pipeline_win2k
readerSMB4=windows,10.0.0.4,Administrator,password,sec:sys,pipeline_win2k_LOW
readerSMB5=windows,10.0.0.5,Administrator,password,sec:sys,pipeline_win2k_LOW
readerSMB6=windows,10.0.0.6,Administrator,password,sec:sys,pipeline_win2k_LOW
readerSMB7=windows,10.0.0.7,Administrator,password,sec:sys,pipeline_win2008
readerSMB8=windows,10.0.0.8,Administrator,password,sec:sys,pipeline_win2008
readerSMB9=windows,10.0.0.9,Administrator,password,sec:sys,pipeline_win2008
#
#define your writers
writerCollector1=collector,5,50000000000
writerCollector2=collector,5,50000000000,win2008
#
#define your pipelines
pipeline_win2k=1000,1,60,writerCollector1,writerRT
pipeline_win2k_LOW=30000,20,60,writerCollector1
pipeline_win2008=1000,1,60,writerCollector2
```

If you run a HawkEye Retriever with the above configuration, it outputs files like the following in the Collector’s `<LogQueue>/spool` directory:

```
sec-2009-02-01T08:02:07.000000Z.tmp
sys-2009-02-01T08:17:05.000000Z.tmp
win2008_sec-2009-02-01T08:32:22.000000Z.tmp
win2008_sys-2009-02-01T09:17:12.000000Z.tmp
```

NOTE: The lower volume win2k data is in the same file as the high volume win2k data.

Given these settings, you would edit the HawkEye Collector config.xml file to map the PTLs as shown below:

```
<PTL table="sensage_win_sec_event" enabled="1" match="sec\-[^/]+$"
namespace="customer.windows">
<PTL table="sensage_win_sec_event_2008" enabled="1" match="win2008_sec\-[^/]+$"
namespace="customer.windows">
```

Case 4: Windows Event Data and SQL Server Audit Logs

This case assumes that your site has a collection of Windows 2003 and 2008 servers (with the 2008 servers generating far less event data) as well as a number of SQL Server databases and that you want to use the standard configuration to poll the two groups of servers and different applications at different rates.

The configurations below illustrate how to configure the readers, the writer, and two pipelines, as well as three options.

```
#define your readers
readerWin2003_1=windows,10.0.0.1,Administrator,password,sec:sys,pipeline_2003
readerWin2003_2=windows,10.0.0.2,Administrator,password,sec:sys,pipeline_2003
readerWin2k8_1=windows,10.0.0.3,Administrator,password,sec:sys,pipeline_2008
readerWin2k8_2=windows,10.0.0.4,Administrator,password,sec:sys,pipeline_2008
readerSQLServerhostA=sqlserverauditlog,10.0.1.15,bob,bobpassword,1433,getTraceFi
les,30000,sqlserver_events,,/tmp/mytrace.sql,pipeline_sqlserver
readerSQLServerhostA=sqlserverauditlog,10.0.1.15,bob,bobpassword,1433,getTraceFi
les,30000,sqlserver_events,,/tmp/mytrace.sql,pipeline_sqlserver

#define your writers
writerCollector=collector,50,500000
writerCollector2=collector,5,50000000000,win2008
writerCollector3=collector,5,50000000000,sqlserver

#define the three pipelines
pipeline_2003=5000,10,60,CollectorWriter
pipeline_2008=30000,1,60,CollectorWriter2
pipeline_sqlserver=10000,1,60,CollectorWriter3

# remote access
remoteshell=9999,localhostOnly
#logging level
logging=INFO,../logs/winretriever.log
```

TESTING YOUR VALUES

To test your data to determine the most effective settings for pipeline arguments, you should:

1. Determine how many events of each event type each reader machine typically receives each hour; use this information to determine the initial configuration settings. For more information, see "Before Configuring the Pipeline", next.

2. Determine whether your initial configuration settings enable the retriever to fetch all records without polling too frequently or resting for too short a delay period. For more information, see "Checking Your Initial Configuration", on page 83.

Before Configuring the Pipeline

Before you configure the pipeline for the first time, you should determine how many events of each type each reader machine typically receives each hour. To make this determination, check the Windows Event Viewer on each machine.

To use Event Viewer to determine event frequency

1. On each machine, open the Event Viewer from the Windows **start** menu:

```
start > Control Panel > Administrative Tools > Event Viewer
```

2. In the left pane, which displays all security event types, double click each event type to display the history for that event type in the right pane. For example, double click **Application**.

3. In the right pane, examine and count the number of events that occur each hour.

4. Average the number of hourly events for each event type over two or three days.

5. Set the delay and request count variables accordingly.

For example, assume a reader machine receives 1000 records each hour. To determine the number received each minute, divide 1000 by 60, which evaluates to 16+ events each minute. Round this value up to 20 to accommodate those periods when the machine receives more than 1000 records each hour. Because each poll request can return up to 100 records, you could set delay to 5 minutes (300000 milliseconds) to guarantee that you collect all records and allow idle time on each machine.

NOTE: Repeat the above procedure on every reader machine.

Checking Your Initial Configuration

After you configure the pipeline for the first time, run the retriever for one or two days to determine whether your configuration settings enable the retriever to fetch all records without polling too frequently or resting for too short a delay period. To make this determination, query the EDW for the period of time you have been running the retriever.

To query the EDW to verify configuration settings

1. Query the EDW to return the average number of events returned by each reader machine for each hour since you began running the retriever.

2. Depending on the volume of data returned by each machine, divide the average events per hour by 60 or 3600 for each machine to determine the average number of events returned each minute or second, respectively.

3. Use the calculated value to determine your request count for high-volume and low-volume machines.

Tuning SMB Event Retrieval

A well-tuned set of pipeline values is one that prevents the reader from excessive polling and yet allows the reader enough polling frequency to keep up with logging. The choice for setting values is affected by three criteria:

- The polling interval and the number of records that are polled at each interval.
- The size of the event log on the Windows host itself.

- The pattern of generation on the Windows host.

In the case of those Windows hosts that generate a steady stream of events, make sure the reader polls often enough so that the reader does not over time fall behind in logging.

It is also possible for Windows hosts to have a workload in which a small trickle of events occur throughout the day, but a large number of events get generated in a short period. For example, a server may generate 20-30 events per second throughout the day, but generate several hundred thousand events during a small ten-minute window.

Tuning then requires looking at the reader’s polling interval and the size of the log on the Windows host. In this case, set the polling interval so that over time the reader can keep up with the total number of events generated per day. In addition, the Windows System Administrator should ensure that the Windows event log has the capacity to hold all the events from that busy ten-minute window and that the time setting is adequate for the HawkEye Retriever to catch up.

The best way to determine the load of a Windows host over time is to measure it. You can check the data returned from pingbox; by issuing multiple calls to pingbox at regular intervals, you can collect data that shows the load (data) that is being generated over time. Based on the load measurement, you can refine your pipeline settings as appropriate.

Tuning Non-SMB Readers

To tune non-SMB readers, modify the polling interval with a setting large enough to accommodate the polling of systems that are not generating data, but also small enough to prevent any poll from pulling millions of event or records. In this case, the correct value is dependent on the type of reader you are using, which determines the amount of data generated. Unlike SMB readers, non-SMB readers do not normally lose data when they fall behind, but you will still want to make sure the reader polls often enough to not fall behind over time.

Tuning SQL Server Readers

The following query provides a rough estimate of how many events per second the server is polling over a somewhat narrow time range:

```
select count(startTime) as recordCount, max(startTime) as maxTime,
min(startTime) as minTime from FROM
sys.fn_trace_gettable('c:\trace\mytrace.trc',1) where startTime is not null;
```

You can sample a couple of different **.trc** files if you know how many **.trc** files there are supposed to be and get the current **.trc** file from this query:

```
SELECT traceid, property, value FROM ::fn_trace_getinfo(null);
```

For example, if the current **.trc** file is **`c:\trace\mytrace_100.trc`**, re-run the initial query above for a couple other samples:

```
select count(startTime) as recordCount, max(startTime) as maxTime,
min(startTime) as minTime FROM
sys.fn_trace_gettable('c:\trace\mytrace_98.trc',1) where startTime is not
null;
select count(startTime) as recordCount, max(startTime) as maxTime,
min(startTime) as minTime FROM
sys.fn_trace_gettable('c:\trace\mytrace_88.trc',1) where startTime is not
null;
```

Defining Remote Shell Access

HawkEye AP provides a remote shell interface to manage the HawkEye Retriever. The remote shell accepts connections from any host through a separate shell process. It connects to the running HawkEye Retriever process over a TCP socket.

- You can run the remote shell from a different host from the one on which the HawkEye Retriever runs, which provides remote administration capabilities.

  Because the remote shell runs in a different process space (and possibly on a different host), it can connect to a HawkEye Retriever that is running as a background process.

- Alternately, you can configure the retriever to allow connections only from the local host.

The syntax for the remote shell is:

```
remoteshell=<port>[,localhostOnly]
```

NOTE: For security purposes, both property files provide the following default for this property:

```
remoteshell=9999,localhostOnly
```

For documentation of all shell commands, see “Shell Commands”, on page 100.

Defining the Logging Level

For each HawkEye Retriever, you can specify the logging level and log file. The default logging level is `INFO`.

The syntax for a logging level is:

```
logging={TRACE|DEBUG|INFO|WARN|ERROR|FATAL},<log_file>
```

The syntax lists the logging levels in the order from most verbose to least verbose.

NOTE: The `agentless.prop` file provides the following default for this property:

```
logging=INFO,<HawkEye AP Home>/var/log/collector/winretriever.log
```

NOTE: The `agent.prop` file provides the following default for this property:

```
logging=INFO,../logs/winretriever.log
```

NOTE: You can use the HawkEye Retriever shell to change the log level at run time. For example, you can increase the log level to a more verbose setting if the log begins to contain errors. For more information, see “Shells”, on page 98.

Defining Non-Standard Log Types

In addition to the five basic log types - Security, Application, System, DSN Server, FIle Replication Service and Directory Service - a Windows host may have other application-specific logs you wish to collect and analyze. On the Windows host these would show up as a unique type in the event server.

To collect these logs you must define a custom log type. In agentless prop, you add a line for each custom log type, mapping it to an abbreviated name. To specify abbreviates for Transform and Windows Powershell logs, you would add the following lines:

```
logmap.transform=Transform
logmap.powershell=Windows Powershell
```

The general specification for the logmap entry is:

```
logmap.<abbreviated name>=Full Log Name
```

NOTE: The abbreviated name can contain only characters and the full name should not be placed in double quotes.

Once you have defined the custom log may entry you can use it just like the other log types in a reader definition. For example, to use the transform log type specified above you could create a reader definition like:

```
readerhost1=windows,1.2.3.4,Administrator,password,sec:transform
```

This would create a reader that will collect both security (sec) and transform data.

Defining the State Files

The `agent.prop` file contains a section on defining the state files. Do not change this setting.

Controlling Historical Data

By default, when a Pipeline is first created, or any time it needs to be reset due to an issue on the Windows host, the Retriever starts pulling the oldest data on the server and loading it. In some cases, this causes a large amount of unwanted historical data to be loaded into the EDW. To prevent this loading, you can specify the following flags in the `agentless.prop` file to control the flow of historical data:

- `event.reader.startnewreadersfromlatest=true`

  Specifies that when a retriever creates a pipeline for the first time, it should *not* retrieve historical data, but instead start retrieving data from the most recent event on the Windows server. This flag only impacts the pipeline when it is first created. After that time, the pipeline defaults to normal behavior and retrieves data from wherever it left off. Setting this flag prevents a new installation from retrieving a large amount of possibly irrelevant historical data.

- `event.reader.startallreadersfromlatest=true`

  Specifies that when a retriever creates a pipeline, or when the Pipeline is restarted (either due to a Collector restart, a `windowsretrievershell` command to reset the pipeline, or when the retriever must restart the pipeline due to an error) it ignores historical data and starts retrieving data from the most recent event.

  This flag affects Pipelines in an ongoing fashion (and therefore could cause data to be lost) and should be used with caution.

Configuring the Agentless Retriever

The procedure below documents how to configure the agentless version of the HawkEye Retriever.

To configure an agentless retriever

1. Change to the directory that contains the `agentless.prop` file.

This file is in the same directory as the Collector `config.xml` file. The default location is:

```
<HawkEye AP Home>/etc/collector/
```

2. Edit the `agentless.prop` file, as described above in "Attributes for HawkEye Retrievers", on page 65.

3. Edit the `config.xml` file to name and define the retriever and point to its properties file.

For information on defining the retriever in the Collector configuration file, see "HawkEye Retriever on Windows", on page 34.

4. Configure authentication against a Windows system for agentless event-log retrieval that uses IPC.

For information, see "Configuring Authentication", on page 103.

Configuring the Retriever to Run as an Agent

The agent version of the HawkEye Retriever has its own archive file: `WindowsRetriever.zip.`

To install and run the agent version of the retriever, you must download t he file from the Hexis Cyber Solutions Technical Support web site:

http://www.hexiscyber.com/content/

IMPORTANT: Before you start the agent version of the HawkEye Retriever for the first time:

- Configure its properties; for more information, see "Attributes for HawkEye Retrievers", on page 65 above.
- Check the Release Notes or PAM for the specific version of Java used in this release of HawkEye AP and add its `bin` directory to your path. To download the JRE:

  ```
  http://java.sun.com/javase/downloads/index.jsp
  ```

To install, configure, run, and test on Windows

1. Download and unzip `WindowsRetriever.zip`.

Unzipping the file creates a new folder named `WindowsRetriever`. This folder is directly below the folder from which you unzipped the file. This folder contains subfolders that include `bin` and `conf`.

2. Change to the `WindowsRetriever\conf` folder and edit the `agent.prop` file, as documented above in "Attributes for HawkEye Retrievers", on page 65.

3. Install the agent as a service:

a Change to the `WindowsRetriever\bin` folder.

b Run:

```
InstallWinRetriever.bat
```

4. Verify that the service was installed correctly by opening:

```
Control Panel > Administrative Tools > Services
```

There should be an installed service named `HawkEye AP Windows Event Retriever`. This service starts automatically the next time Windows is started. However, you can start it at any time by right-clicking the service and selecting **Start**.

5. Verify the installation by running the following command from the `WindowsRetriever\bin\` folder:

```
windowsretrievershell.exe localhost 9999
```

NOTE: After you have invoked the shell, you can run `help` to list all available commands. For a complete list of shell commands, see "Shell Commands", on page 100.

ADVANCED CONFIGURATION

Generally you configure the HawkEye Retriever and its submodules by editing the standard properties in the `agentless.prop` or `agent.prop` file. However, you may need to define a HawkEye Retriever with more than the basic functionality enabled by the default standard configuration. HawkEye AP provides a full set of options that allow more flexibility when you configure a HawkEye Retriever. For example, you can use the advanced options to:

- chain one HawkEye Retriever to another
- automatically discover Windows hosts from a directory service
- run an interactive shell for debugging purposes
- specify a non-default format and string separator for timestamps; the default is: `yyyy-MM-dd HH:mm:ss`
- specify retry value for the TCP event writer

To enable any of the above advanced options, you must replace the standard syntax for that option in the `agentless.prop` or `agent.prop` file with the full set of advanced properties that define the desired submodule. Additionally, you must specify the full set of advanced properties that define all other submodules. In other words, to take advantage of any advanced option, you must copy the full set of advanced properties from this chapter and paste them into the `agentless.prop` or `agent.prop` file.

Examples:

- For an example of how you would modify the property file to configure a flat-file event writer to overwrite rather than append to the output file, see "Introduction", on page 65.
- For an example of an advanced agentless.prop file, see "Example of Advanced Properties File", on page 108.

The sections below document the full set of options for the HawkEye Retriever.

- "Event Readers", on page 89
- "Event Writers", on page 94
- "Serializers", on page 97
- "Pipelines", on page 97
- "Shells", on page 98
- "Discoverers", on page 101

- “Logging”, on page 102

Event Readers

Every HawkEye Retriever must have at least one event reader. Each event reader specifies the source of event data, which can be any of the following:

- Windows SMB connections
- Oracle Audit or generic table data (BETA)
- SQL Server generic table data
- SQL Server Audit Trace data
- SQL Server 2000 Audit Trace data
- Flat files
- Multiple flat files
- TCP sockets

Each event reader maps to exactly one source of Windows event data. Hexis Cyber Solutions recommends that you name each reader to indicate the type of data the reader collects; for example: `smbReader`, `flatFileReader`, and `tcpReader`. Names are case sensitive. Unlike a standard-configuration reader, there is no restriction on how you name the reader. For example, `readerSMB`, `ReaderSMB`, and `SMBreader` are all valid.

This property is required.

If you do not want newly created readers to pull historical data, you can set a flag that affects all new readers. Set this flag to cause a new reader to start pulling over new records, but not large volumes of historical data. This flag can also help prevent pulling duplicate data in the event you migrate readers across collectors or have lost the state database.

```
event.reader.startfromlatest=true;
```

To define an event reader

1. Declare the readers in the `event.readers` property; for example:

```
event.readers=smbReader,smbReaderLow,smbReaderHigh,flatFileReader,tcpReader
```

2. Specify the class in the `event.reader.<`*`reader_name`*`>.class` property. Acceptable values are:

- `com.sensage.eventlog.reader.SMBEventReader`
- `com.sensage.eventlog.reader.FileEventReader`
- `com.sensage.eventlog.reader.TCPEventReader`

The specified class determines the remaining properties to set.

Defining an SMB Event Reader

Properties unique to this reader are:

- **host**—Specify the IP address or WINS name of the Windows host from which to connect using SMB.
- **username**–Specify the user name for SMB authentication.
- **password**—Specify the password for SMB authentication.
- **eventtypes** —See the section “Setting an Event Type”, on page 74 for more information.

NOTE: The SMB event reader reads event-log records from a remote SMB session that uses DCE/RPC.

Defining an Oracle Audit or Generic Table Reader (BETA)

Properties unique to this reader are:

- **hostname**—The IP address or FQDN of the Oracle host
- **username**—The Oracle account used to access the table (must have Select permissions to that table)
- **password**—The password supplied for this user.
- **port**—The port used to contact the Oracle host.
- **servicename**—Specify the Oracle service name.
- **namedsqlstring**—References a string in the sqlstringmap to use to query the server.
- **eventtype**—See the section “Setting an Event Type”, on page 74 for more information.
- **passwordmanager**—The name of a defined password manager that will be used for this reader.
- **daysprevious**—A number (0 or higher) that indicates how many days in the past to start pulling data the first time this reader is used.
- **tscol**—The column name of a column that has a timestamp the retriever can use to track data it has already read.
- **timestampformat**—A String that specifies a non-standard timestamp format, the default is "yyyy-mm-dd hh24:mi:ss.ff"

NOTE: There are two additional properties that affect all Oracle readers in the instance: event.reader.oraclejdbcclass defaults to com.ddtek.jdbc.oracle.OracleDriver but could be modified to use other JDBC drivers in some instances **event.reader.oraclejdbcur** defaults to `jdbc:datadirect:oracle://HOSTNAME:PORT;ServiceName=SERVICENAME` but could be modified to support other connection URLs.

Defining a SQL Server Generic Table Reader

Properties unique to this reader are:

- **hostname**—The IP address or FQDN of the SQL Server host.
- **username**—The SQL Server account used to access the (must have Select permissions to that table)
- **password**—The password supplied for this user.
- **port**—The port used to contact the SQL Server host.
- **databasename**—The database on SQL Server that you are accessing.

- **eventtype**—See the section “Setting an Event Type”, on page 74 for more information.
- **tscol**—The column name of a column that has a timestamp the retriever can use to track data it has already read.
- **namedsqlstring**—References a string in the sqlstringmap to use to query the server.

Defining a SQL Server Audit Trace Reader

Properties unique to this reader are:

- **hostname**—The IP address or FQDN of the SQL Server host.
- **username**—The SQL Server account used to access the (must have Select permissions to that table)
- **password**—The password supplied for this user.
- **port**—The port used to contact the SQL Server host.
- **trace definition file**—A file name that points to a file containing a trace definition.
- **metaDataFunctionName**—The name of a function on the server that returns metadata about the **.trc** files.
- **max records per poll**—The most records that can be brought back in a single polling interval. The default is 25000.
- **eventtype**—See the section “Setting an Event Type”, on page 74 for more information.
- **tscol**—An optional column name for the column in the trace that has a valid timestamp. This is normally not set.

Defining a SQL 2000 Server Audit Trace Reader

NOTE: This is for remote agents on a SQL Server host).

Properties unique to this reader are:

- **hostname**—The IP address or FQDN of the SQL Server host.
- **username**—The SQL Server account used to access the (must have Select permissions to that table)
- **password**—The password supplied for this user.
- **port**—The port used to contact the SQL Server host.
- **tracefile**—Path to a file that defines the SQL Server Trace.
- **max records per poll**—The most records that can be brought back in a a single polling interval. The default is 25000.
- **eventtype**—See the section “Setting an Event Type”, on page 74 for more information.
- **tscol**—The column name of a column that has a timestamp the retriever can use to track data it has already read.

Defining a File Event Reader

Required properties unique to this reader are:

- **filename**—The flat file from which to read records.
- **blocksize**—The number of records to read in each pass.
- **hashsize**—The number of bytes used to generate a unique hash on the file; allow for enough bytes for each file to generate a unique set of bits. The default is 2048.
- **maximum-noreads**—How many times the retriever can read no new lines from a file before it attempts to look for log rolls.
- **hashsize**—How many bytes are used from the beginning of a file to determine if a log has rolled.
- **eventype**—A string that is prepended to every record.
- **prepend**—A value of true or false to indicate that the filename should be prepended to every record.

Defining a Multifile Event Reader

Required properties unique to this reader are:

- **searchString**—The directories and files from which to read records.

Optional properties unique to this reader are:

- **blocksize**—The number of records to read in each pass.
- **minutesbetweenrefresh**—The number of minutes the multifile reader waits before it looks for new matching directories.
- **hashsize**—The number of bytes used to generate a unique hash on the file; allow for enough bytes for each file to generate a unique set of bits. The default is 2048.
- **maximum_noreads**—The number of times the reader can read 0 new bytes before checking to see if a file has rolled or new matching files have arrived.

Define this type of event reader when you want to define a single reader to scan multiple files in different directories in parallel (Microsoft IIS is a good example). Assume, for instance, that you have websites that log a unique log file per website in a directory hierarchy as in:

```
/var/log/websites/<web server name>/logs/<server name>.log
```

and those log files roll out to something like:

```
/var/log/websites/<web server name>/logs/<server name>.log.yyyy.mm.dd
```

then you would specify a multifile reader as in:

```
reader_myMultiReader=multifile,/var/log/websites//logs/.log*
```

This creates a reader that will, in parallel, monitor any directory matching the search expression and read log files as they are written. The multifile reader can detect (inside any matching directory) if a log file was renamed so that it will not collect the data twice.

By default the string assumes every character is literal except you can use a **?** (question mark) to indicate a single character and an * (asterisk) to indicate multiple characters (for example: **/tmp/dir?/logifle*.log)**

Specify pathnames using the appropriate path separator for that operating system. For example on windows specify a drive letter and use the backslash, **c:\\tempfile**, and on Linux you use the forward slash, **/tmp/logfile.out**

Optionally, you can use a regex to define your search string. To do this you must first set this global variable to true:

```
event.readers.useregexforfilesearch=true
```

Then your search string can be specified using a valid java regex for example:

```
/var/log/myapp/day??/[Ll]ogF(l|ile).*\.log)
```

The only rule is that the regex must include a path separator between the directories to scan and which files to scan for. The retriever will use that path separator to determine which directories to scan versus which files to scan.

Performance Details and Recommendations for Configuration

For Multifile Retriever, recommendations were based on a test using the following values:

- Maximum file size tested: 2 MB
- Maximum number of files tested: 1500 files
- The "avg" file size tested for "n" files retrieved in time "t" was performed on different file size log files: <1MB, 1MB, 2MB

Multiile Retriever has been tested on 1500 files and up to 7-nested directory levels.

Defining a TCP Event Reader

Properties unique to this reader are:

- **port—**Specify the TCP port on which to listen for incoming connections from TCP Event Writers.

Define this type of event reader to chain one HawkEye Retriever to another. For example, when you run the HawkEye Retriever as an agent on Windows, you must configure it to forward its data either to a HawkEye Retriever that runs on a Linux system or to a HawkEye AP Syslog Writer that runs on a Linux host. You must define a TCP event reader to forward the retriever agent on Windows to the agentless retriever on Linux.

Example: Defining Event Readers

The example below illustrates configuration of a retriever that collects event data from all three of the possible data sources. The first section of the file declares all three readers.

```
# Declare the set of readers
event.readers=smbReader,flatFileReader,tcpReader,tcpReaderLow,tcpReaderHigh
# Define the smbReader
event.reader.smbReader.class=com.sensage.eventlog.reader.SMBEventReader
event.reader.smbReader.eventtypes=Security,Application,System,DNS Server,File
Replication Service,Directory Service
event.reader.smbReader.host=127.0.0.1
event.reader.smbReader.username=administrator
event.reader.smbReader.password=mypass
# Define the flatFileReader
event.reader.myflatFileReader.class=com.sensage.eventlog.reader.FileEventReader
event.reader.myflatFileReader.filename=/<full_path>/sample.log
event.reader.myflatFileReader.requestcount=1000
event.reader.myflatFileReader.maximum_noreads=5
event.reader.myflatFileReader.hashsize=2048
event.reader.myflatFileReader.eventype=sample
event.reader.myflatFileReader.prepend=true
# Define the tcpReaders
event.reader.tcpReader.class=com.sensage.eventlog.reader.TCPEventReader
event.reader.tcpReader.port=8099
event.reader.tcpReaderLow.class=com.sensage.eventlog.reader.TCPEventReader
event.reader.tcpReaderLow.port=8099
event.reader.tcpReaderHigh.class=com.sensage.eventlog.reader.TCPEventReader
event.reader.tcpReaderHigh.port=8099
```

Event Writers

When you use advanced configuration options, ensure that each HawkEye Retriever has at least one event writer. Each event writer specifies the destination of event data.

When you run the HawkEye Retriever on Windows, you can define a event writer to send the data to one of three destination types:

- Flat file
- TCP socket for chaining to another HawkEye Retriever
- syslog writer

The event writer has a different set of properties for the different destination types.

When you run the HawkEye Retriever on Linux, you can define a event writer to send the data to the Collector log queue for batch collection

Each event writer maps to exactly one destination for Windows event data and uses an event serializer to format each event record to a string. For more information, see "Serializers", on page 97.

To define an event writer

1. Declare the event writer in the `event.writers` property; for example:

```
event.writers=collectorWriter
```

NOTE: Unlike a standard-configuration writer, there is no restriction on how you name the writer. For example, `collectorWriter`, `CollectorWriter`, and `writerCollector` are all valid

2. Specify its class in the `event.writer.<writer_name>.class` property. Acceptable values are:

- `com.sensage.eventlog.writer.CollectorEventWriter`
- `com.sensage.eventlog.writer.RTEventWriter`
- `com.sensage.eventlog.writer.TCPEventWriter`

- `com.sensage.eventlog.writer.FileEventWriter`
- `com.sensage.eventlog.writer.SyslogEventWriter`
- `com.sensage.eventlog.writer.TCPSyslogEventWriter`

The specified class determines the remaining properties to set.

Defining a Collector Event Writer

Properties unique to this event writer are:

- **threshold.filesize** —Specify the maximum size of the log queue file in bytes before it is renamed and collected.
- **threshold.minutes** —Specify the maximum age of the log queue file in minutes before it is renamed and collected.
- **filename.prefix** —Specify a prefix to be added at the beginning of the name of the output files for this writer, and configure the Collector to look for files with this name.

This event writer writes event-log records to Collector files in a log queue. It is used only when retrieving batched data. The example below configures a retriever that forwards event data to the Collector log queue for batch processing.

```
# Declare the writer
event.writers=collectorWriter
# Define the collectorWriter
event.writer.collectorWriter.class=com.sensage.eventlog.writer.CollectorEventWri
ter
event.writer.collectorWriter.threshold.filesize=50000000
event.writer.collectorWriter.threshold.minutes=60
event.writer.collectorWriter.serializer.class=com.sensage.eventlog.serializer.De
limitedSerializer
event.writer.collectorWriter.serializer.delimiter=|
event.writer.collectorWriter.serializer.date.format=yyyy-MM-dd HH:mm:ss
event.writer.collectorWriter.filename.prefix=win2k8
```

Defining a TCP Event Writer

Properties unique to this event writer are:

- **host**— IP address or resolveable hostname where there is a listening TCP event reader.
- **port**—Port number used by the listening TCP event reader.
- **retry**—Milliseconds to wait before retrying to open the TCP connection if the connection is lost.

NOTE:
(1) This configuration is useful for chaining one HawkEye Retriever to another. For an illustration of chaining two HawkEye Retriever see “HawkEye Retriever on Windows”, on page 34.
(2) This writer is also used to send data to the Real-Time system. In this case you would specify the hostname and IP address of the real time parser.

The example below configures a retriever that forwards event data to a TCP socket for chaining to another HawkEye Retriever.

```
# Declare the writer
```

```
event.writers=tcpWriter
# Define the tcpWriter
event.writer.tcpWriter.class=com.sensage.eventlog.writer.TCPEventWriter
event.writer.tcpWriter.host=127.0.0.1
event.writer.tcpWriter.port=8099
event.writer.tcpWriter.retry=2000
event.writer.tcpWriter.serializer.class=com.sensage.eventlog.serializer.Delimite
dSerializer
```

Defining a File Event Writer

Properties unique to this event writer are:

- **filename** —Name of the output file.
- **append**—Set to `true` to append to this file; set to `false` to overwrite this file.

The example below configures a retriever that forwards event data to a flat file.

```
# Declare the writer
event.writers=fileWriter
# Define the fileWriter
event.writer.fileWriter.class=com.sensage.eventlog.writer.FileEventWriter
event.writer.fileWriter.filename=/<full_path>/output.log
event.writer.fileWriter.append=true
event.writer.fileWriter.serializer.class=com.sensage.eventlog.serializer.Delimit
edSerializer
```

Defining a UDP Syslog Event Writer

Properties unique to this event writer are:

- **host**—IP address or resolveable host name of the listening syslog daemon.
- **port**— Port of the listening syslog daemon.

The example below configures a retriever that forwards event data to the HawkEye AP Syslog Writer over UDP.

```
# Declare the writer
event.writers=syslogUDPWriter
# Define the syslogUDPWriter
event.writer.syslogUDPWriter.class=com.sensage.eventlog.writer.SyslogEventWriter
event.writer.syslogUDPWriter.host=127.0.0.1
event.writer.syslogUDPWriter.port=514
event.writer.syslogUDPWriter.serializer.class=com.sensage.eventlog.serializer.De
limitedSerializer
```

Defining a TCP Syslog Event Writer

Properties unique to this event writer are:

- **host**—IP address or resolveable host name of the listening syslog daemon.
- **port**— Port of the listening syslog daemon.

The example below configures a retriever that forwards event data to the HawkEye AP Syslog Writer over TCP.

```
# Declare the writer
event.writers=syslogTCPWriter
# Define the syslogTCPWriter
event.writer.syslogTCPWriter.class=com.sensage.eventlog.writer.TCPSyslogEventWri
ter
event.writer.syslogTCPWriter.host=127.0.0.1
event.writer.syslogTCPWriter.port=514
event.writer.syslogTCPWriter.serializer.class=com.sensage.eventlog.serializer.De
limitedSerializer
```

Serializers

Serializers control the formatting of an event record into a string to be written. Event writers use the serializer every time they write an event record to its destination as a string. The standard serializer is the Delimited Serializer, which writes event records as a string of delimited fields.

Properties unique to the `DelimitedSerializer` serializer are:

- **class**—Full Java class of the serializer implementation

  This property is required to tell the event writer which serializer to use.

- **delimiter**—String representing the delimiter to add between each field.

- **date.format**—Date formatting string that controls the normalizing of timestamps

  For more information, see:

  `http://java.sun.com/j2se/1.5.0/docs/api/java/text/SimpleDateFormat.html`

Example: Using a Serializer

The example below illustrates configuration of a Collector event writer that specifies a pipe (`|`) as the serializer delimiter and formats the timestamp to begin with year.

```
# Define the collectorWriter
event.writer.collectorWriter.class=com.sensage.eventlog.writer.CollectorEventWri
ter
event.writer.collectorWriter.threshold.filesize=50000000
event.writer.collectorWriter.threshold.minutes=60
event.writer.collectorWriter.serializer.class=com.sensage.eventlog.serializer.De
limitedSerializer
event.writer.collectorWriter.serializer.delimiter=|
event.writer.collectorWriter.serializer.date.format=yyyy-MM-dd HH:mm:ss
```

Pipelines

Although the standard configuration for a single pipeline type automatically identifies each pipeline’s reader and writer(s), you must specify the writer(s) and the pipeline name when you use standard configuration to define multiple pipeline types. Then you must configure each reader to use a specific pipeline type. Event pipelines are documented in detail in “Defining a Pipeline”, on page 76.

Although you can use standard configuration to create pipelines with different values, you must use advanced configuration to create pipelines that filter the records separately for each reader before they are written.

Advanced pipeline properties are:

- **reader**—named event reader that is unique to the pipeline
- **writer**—named event writer that can be shared among pipelines
- **delay**—specified in milliseconds, represents the amount of time that the retriever waits before polling each reader for additional event records; use this argument to limit the load on the each reader. Defaults to `1000`. For more information, see "Defining Delay and Request Count", on page 77.
- **requestcount**—specified as an integer, represents the maximum number of times the retriever polls each reader before the delay period begins; regardless of the number of records in the reader's buffer, each poll request returns a maximum of 100 event records. Defaults to `1`.
- **zerocounter**—specified as an integer, represents the maximum number of times a reader can return zero (0) records before the HawkEye Retriever logs a warning in the log file indicating that the retriever might have fallen behind in gathering records. Defaults to `60`.

  NOTE: The zero counter is used by the database retriever (SQL Retriever) to check the trace status on the SQL Server and to attempt to start the trace if it has stopped. For more information, see "Defining the Zero Counter", on page 79.

Example: Defining 2 Pipelines

The example below illustrates the advanced configuration properties to define two pipelines. Each pipeline connects an SMB event reader with a writer: the Collector. One pipeline is configured for readers that regularly receive a significant number of events. The other pipeline is configured for readers that do not receive a significant number of events.

```
# Declare 2 pipelines
event.pipelines=HighVolumePipeline,LowVolumePipeline
# Define the high-volume pipeline
# Defines pipeline for an SMB reader that reads only Security & Application logs
event.pipeline.HighVolumePipeline.reader=smbReaderSecAppLogs
event.pipeline.HighVolumePipeline.writer=collectorWriter
# Sets the pipeline to delay every second
event.pipeline.HighVolumePipeline.delay=1000
# Sets the pipeline to receive no more than 1000 records each polling period
event.pipeline.HighVolumePipeline.requestcount=10
# Sets the pipeline to check for zero records no more than once a minute
event.pipeline.HighVolumePipeline.zerocounter=60
# Define the low-volume pipeline
# Defines pipeline for an SMB reader that reads all log types
event.pipeline.LowVolumePipeline.reader=smbReaderAllLogs
event.pipeline.LowVolumePipeline.writer=collectorWriter
# Sets the pipeline to delay every 5 minutes
event.pipeline.LowVolumePipeline.delay=300000
# Sets the pipeline to receive no more than 100 records each polling period
event.pipeline.LowVolumePipeline.requestcount=1
# Sets the pipeline to check for zero records every 20 minutes
event.pipeline.LowVolumePipeline.zerocounter=4
```

Shells

HawkEye AP provides a command-line shell interface to manage the HawkEye Retriever There are two different ways to access the shell interface of a running HawkEye Retriever.

- **interactive shell**—runs as part of the retriever process and accepts input from the standard input directly

  This mode is mostly for debugging because you must run the HawkEye Retriever as a foreground process.

- **remote shell**—accepts connections from any host through a separate shell process; connects to the running HawkEye Retrieverprocess over a TCP socket.
  - You can run the remote shell from a different host from the one on which the HawkEye Retriever runs, which provides remote administration capabilities.
  - You can run the remote shell on the same host on which the HawkEye Retriever runs, which allows you to monitor the retriever when it runs as a background process.

Shell properties are:

- **interactive.shell.enabled**—Specify `true` for interactive mode or `false` for daemon mode.
- **remote.shell.enabled**—Specify `true` to allow connecting when in daemon mode from any host.
- **remote.shell.listen.port**—Specify the port on which to listen for remote connections.
- **remote.shell.localhost.only**—Specify `true` to allow only connections from this host or `false` to allow connections from any host.

Invoking the Interactive Shell

Before you invoke the interactive shell, you must set the `interactive.shell.enabled` configuration property to `true`:

```
interactive.shell.enabled=true
```

NOTE: To invoke the interactive shell, you must run the HawkEye Retriever as a foreground process.

Invoking the Remote Shell

Before you invoke the remote shell, you must set the following configuration properties:

```
# Enable remote shell sessions.
remote.shell.enabled=true
# Set port on which the Windows Event Retriever listens for
#    incoming remote-shell sessions.
remote.shell.listen.port=9999
# Set Localhost Only property: true allows only connections from this host;
#   false allows connections from any host.
remote.shell.localhost.only=false
```

To connect to the Windows Event Retriever from a remote shell

Run the following command:

```
/<full_path>/windowsretrievershell <host> <port>
```

For example:

```
bin/windowsretrievershell localhost 9999
```

The command above opens a remote shell on the local host on port 9999.

Example: Defining a Shell

The example below illustrates defining an interactive shell and a remote shell:

```
# Disable the interactive shell interface for daemon mode
interactive.shell.enabled=false
# Configure the remote shell interface to enable remote connections
remote.shell.enabled=true
remote.shell.listen.port=9999
remote.shell.localhost.only=false
```

Shell Commands

After you have invoked the shell, you can run `help` to list all available commands.

Available commands are:

Command	Description
`config [`*`<property_key>`*`]`	Displays the configuration property, or all properties if none specified.
`getloglevel`	Returns the current log level.
`help`	Displays all commands
`pingbox` *`<logtype> <ipaddress> <username> <password>`*	Tests connectivity to the host specified by IP address by using the specified username and password to read records from the specified log type. You can use this command to test machines not currently in `agentless.prop`.
`pingbox` *`<logtype> <readername>`* `[BATCH]`	Tests connectivity to the host specified by readername by reading records from the specified log type. The optional `BATCH` flag returns only PASS/FAIL as opposed to detailed information.
`pingbox` *`<logtype>`* `[BATCH]`	Tests connectivity to all hosts in the `agentless.prop` file. For each host, it attemps to read a single record from the host to ensure complete connectivity. The optional `BATCH` flag returns only a PASS/FAIL as opposed to detailed information for each reader for easy parsing. **Note:** The command does not produce output until all readers have responded and thus may take a while to run.
`setloglevel [VERBOSE|DEBUG|INFO|WARN|ERROR]`	Changes the log level to the specified value until HawkEye Retrieveris restarted.
`shutdown`	Stops all pipelines that are currently running and stops the retriever process.
`start` *`<pipeline>`*	Starts the specified pipeline.
`startall`	Starts all pipelines that are currently stopped.
`state` *`<pipeline>`*	Shows the state information for the handles of the specified pipeline.

Command	Description
`status [<pipeline>|<reader>|<writer>]`	Shows the status of the specified pipeline, event reader, or event writer. Shows all pipelines if none specified.
`stop <pipeline>`	Stops the specified pipeline.
`stopall`	Stops all pipelines that are currently running.
`reloadpropfile CONFIRM`	Reexamines the `agentless.prop` file to determine whether readers have been removed or added. If so: Shuts down the corresponding pipelines for removed readers Creates new pipelines for new readers and starts them
`reset <pipeline>`	Resets the state of the pipeline, which causes the pipeline's handle(s) to start reading from the oldest available record number(s). **WARNING**: This command could adversely affect log retrieval.
`resetall`	Resets the state of all pipelines.
`resetreader <readername> CONFIRM`	Resets the specified reader's oldest record value and current record value for all log types. It effectively ignores the internal state and asks the Windows host for the information. You can use this command to recover from some failure cases.
`quit | q`	Quits this shell.

EXAMPLES

Display all available remote shell commands:

```
help
```

Display the status of the single pipeline:

```
status
```

Display the detailed status of a specific event reader or the event writer:

```
status SMBreader
```

```
status CollectorWriter
```

Exit the shell:

```
quit
```

Discoverers

Discoverers allow you to automatically configure readers and pipelines at runtime. All discoverer types can find Windows hosts for SMB connections.

Using discoverers simplifies the configuration file because you do not need to specify any event readers or pipelines. You need to define only a single event writer, which will be used by the pipelines that are automatically generated for each discovered reader.

The default discoverer is LDAPDiscoverer, which finds hosts by running an LDAP query. This object is useful to read from many Windows hosts that are specified as members of a group in an LDAP-compliant directory service, such as Active Directory.

Example: Defining a Discoverer

The example below illustrates a discoverer configuration that queries LDAP to locate host names that are members of a group `DN`:

```
# Configure auto-discovery
config.discovery.class=com.sensage.eventlog.util.LDAPDiscoverer
config.discovery.username=administrator
config.discovery.password=changeme
config.discovery.eventtypes=Security,Application,System,DNS Server,File
Replication Service,Directory Service
config.discovery.pollinginterval=5000
config.discovery.blocksize=1
config.discovery.ldap.host=10.0.2.143
config.discovery.ldap.port=389
config.discovery.ldap.version=3
config.discovery.ldap.loginDN=ss_enumerator@sensage.local
config.discovery.ldap.password=changeme
config.discovery.ldap.groupDN=cn=adservers,ou=sensage,dc=sensage,dc=local
event.writers=testWriter
event.writer.testWriter.class=com.sensage.eventlog.writer.FileEventWriter
event.writer.testWriter.filename=logs/test.log
event.writer.testWriter.append=true
event.writer.testWriter.serializer.class=com.sensage.eventlog.serializer.Default
Serializer
```

The above example creates all readers as SMBEventReaders that get their properties (`username`, `password`, `eventtypes`, `delay`, `requestcount`) from the `config.discovery` settings. The LDAP connection settings and query are set by the `config.discovery.ldap` properties. This query searches for all members of the `groupDN` and assumes they are valid Windows system names that can be connected to through SMB for event records. All pipelines created share the same event writer.

Logging

The HawkEye Retriever uses Log4j as the logging subsystem. The logging properties you define are passed directly to the Log4j API.

Example: Rolling Over the HawkEye Retriever Log File

The example below illustrates how to pass in a single log4j parameter so that the windows retriever log file is rolled over every month:

```
log4j.appender.LOGFILE.DatePattern='.'yyyy-MM
```

Example: Defining Logging

The example below illustrates a standard configuration that logs to a file with an INFO threshold:

```
log4j.rootCategory=INFO, LOGFILE
log4j.appender.LOGFILE=org.apache.log4j.FileAppender
log4j.appender.LOGFILE.File=logs/retriever.log
log4j.appender.LOGFILE.Append=true
log4j.appender.LOGFILE.Threshold=INFO
log4j.appender.LOGFILE.layout=org.apache.log4j.PatternLayout
log4j.appender.LOGFILE.layout.ConversionPattern=%p %d{ISO8601}: %m%n
```

CONFIGURING AUTHENTICATION

NOTE: The Configuring Authentication Section applies to Windows SMB readers only.

In addition to defining the properties for your HawkEye Retriever, you must configure authentication against a Windows server for event-log retrieval. You can configure authentication using either Administrator or non-Administrator access. There are some general configuration settings that are common to either type of access, discussed next, and there are specific instructions for each type.

- “General Authentication Configuration”, next
- “Administrator Access”, on page 103
- “Non-Administrator Access”, on page 104
- “Forwarded Events Log in Windows 2008”, on page 106

General Authentication Configuration

Configure the following in addition to configuring Administrator or non-Administrator access:

- The account used to authenticate must be granted the "Access this computer from the network" user right. This may be accomplished explicitly or inherently if the Users or Domain user group is granted this right.
- The host running the HawkEye Retriever must be able to open the `IPC$` share on the remote Windows server.
- The Windows device and remote Windows server must be able to communicate over SMB port 445.
- The event log service must be on.
- In Windows 2008, the firewall must allow File and Print sharing.
- There are additional Log Adapter setup instructions for the `microsoft_windows_securityEvent_sensageRetriever` Log Adapter.
  See Setting up the microsoft_windows_securityEvent_sensageRetriever Log Adapter in Chapter 22, “microsoft_windows_securityEvent_sensageRetriever,” in the *Analytics Guide*.

Administrator Access

Create a user that is a member of `Administrators` or `Domain Administrators`. This approach is the easiest to implement and is a common approach for agentless Windows event-log collection.

IMPORTANT: (Windows 2008 only) This user must be assigned to the **Event Log Readers** group.

Non-Administrator Access

NOTE: These instructions assume you have a domain user with non-administrator access. If such a user does not exist, create it.

There are three high-level steps to granting non-administrator access:

1. Get the Security ID (SID) of the domain user.

2. Set the Group Policy Editor (`gpedit.msc`) to display log-policy security options.

NOTE: You need to perform this step once on any host in the domain.

IMPORTANT: If you need to run **gpedit.msc** on a Windows 2008 Server, create a registry key. See “Delegating access to an event log using the registry”, on page 106.

3. Edit the security descriptor for each event-log that you want to monitor.

NOTE: You edit the security descriptor only on the machine whose Group Policy Editor you are using or modifying in Step 2.. You do not edit the descriptor on every host in your domain; the changes are automatically pushed to all domain hosts.

IMPORTANT: Contact Hexis Cyber Solutions Technical Support before you perform this procedure.

To get the SID of the domain user

1. From a command prompt, run:

```
wmic
```

2. Type:

```
useraccount where name='<user name>' get sid
```

The SID displays.

NOTE: An alternate way to get the SID is to run `getsid.exe`.

To set the Group Policy Editor to display event-log security options

1. Use a text editor such as Notepad to open `Sceregvl.inf` in the `%Windir%\Inf` folder.

2. Add the lines below to the `[Register Registry Values]` section.

IMPORTANT: Each of the six lines below begins with the word "`MACHINE`". All text that wraps to a new line must be entered on the same line that begins with "`MACHINE`". The last three lines wrap at a space. You must enter a space in these folder names: "`Directory Service`", "`DNS Server`", and "`File Replication Service`".

```
MACHINE\System\CurrentControlSet\Services\Eventlog\Application\CustomSD,1,%App
CustomSD%,2
MACHINE\System\CurrentControlSet\Services\Eventlog\Security\CustomSD,1,%SecCus
tomSD%,2
MACHINE\System\CurrentControlSet\Services\Eventlog\System\CustomSD,1,%SysCusto
mSD%,2
MACHINE\System\CurrentControlSet\Services\Eventlog\Directory
Service\CustomSD,1,%DSCustomSD%,2
MACHINE\System\CurrentControlSet\Services\Eventlog\DNS
Server\CustomSD,1,%DNSCustomSD%,2
MACHINE\System\CurrentControlSet\Services\Eventlog\File Replication
Service\CustomSD,1,%FRSCustomSD%,2
```

3. Add the lines below to the [Strings] section.

IMPORTANT: Enter the following six lines; do not enter wrapped text on a separate line.

```
AppCustomSD="Eventlog: Security descriptor for Application event log"
SecCustomSD="Eventlog: Security descriptor for Security event log"
SysCustomSD="Eventlog: Security descriptor for System event log"
DSCustomSD="Eventlog: Security descriptor for Directory Service event log"
DNSCustomSD="Eventlog: Security descriptor for DNS Server event log"
FRSCustomSD="Eventlog: Security descriptor for File Replication Service event
log"
```

4. Save your changes.

5. Run the following command:

```
regsvr32 scecli.dll
```

To edit the security descriptor for an event log

1. Run `gpedit.msc`.

2. Expand the following:

`Local Computer Policy` > **`Computer Configuration`** > **`Windows Settings`** > **`Security Settings`** > **`Local Policies`** > **`Security Options`**

You should see the security descriptors that you defined in Step 3. of "To set the Group Policy Editor to display event-log security options", on page 104.

3. Add the following to the end of each event-log security descriptor that you want to monitor:

```
O:BAG:SYD:(A;;0x1;;;<SID>)
```

where `<SID>` is the Security ID of the specific user.

The above setting tells each event log to *accept* `(A)` requests from the *specified user*`(<SID>)` to *read* `(0x1)` from the event log.

Delegating access to an event log using the registry

Use the procedure below if you need to run **gpedit.msc** on a Windows 2008 Server, in which case you need to create a registry key for it.

IMPORTANT: In case of incorrect edits, back up any valued data on the computer before making changes to the registry.

NOTE: The procedure actually changes the SDDL (Security Descriptor Definition Language) and unlike XP/Win2003, does not make visible a string in the REG_SZ value called "CustomSD" . After this procedure, you will need to take the registry key changes, export them, and import them to the machines or use GPO (Group Policy Object) to deploy the registry changes.

This procedure is from the following site: http://technet.microsoft.com/en-us/library/cc722385(WS.10).aspx

1. Open Registry Editor.

2. Navigate to the following registry path:

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\EventLog

You will see that there are keys available for each event log. Select the event log for which you want to delegate read-only access.

3. Add a new key with the name CustomSD to the event log you selected.

4. Add a new String value to the CustomSD key. The of this string is not required, but it represents the access control list for the event log in the Security Descriptor Definition Language (SDDL) syntax. In this procedure this value will be refreed to as *SDDLACL*.

5. Set the value of *SDDLACL* to the following*:*

```
O:BAG:SYD:(D;;0xf0007;;;AN)(D;;0xf0007;;;BG)(A;;0xf0007;;;SY)(A;;0x7;;;BA)(A;;0x
5;;;SO)(A;;0x1;;;IU)(A;;0x1;;;SU) (A;;0x1;;;S-1-5-3)(A;;0x2;;;LS)(A;;0x2;;;NS)
```

After yu edit this value and restart the computer, the new setting will take effect.

Forwarded Events Log in Windows 2008

Windows 2008 introduced a new log type called the *Fowarded Events Log*. This log is used by Windows 2008 to collect logs from a remote host onto one centralized windows server. To collect and load the Forwarded Events Log, perform these steps:

1. Add a new log type in your properties file. For example:

```
logmap.fwd=ForwardedEvents
```

2. Provide the new log type in your reader definitions. For example:

```
reader1=windows, 1.2.3.4,Administrator,password,sec:fwd
```

NOTE: The above example is used to gather both the Security log and the Forwarded Events Log.

3. On the new windows server itself, perform the following steps:

a Run the Registry Editor.

b If the key name does not already exist, add **Forwarded Events** under `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\eventlog`.

IMPORTANT: Be sure to separate the two words of the key name **Forwarded Events** with a space.

c Add the key name **DisplayNameFile** under `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\eventlog\Forwarded Events` of type `REG_EXPAND_SZ`. The value of this key should match the value in `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\eventlog\Application\DisplayNameFile`.

d Add the key name **File** under `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\eventlog\Forwarded Events` of type `REG_EXPAND_SZ`. The value of this key should be **%systemroot%\system32\winevt\Logs\ForwardedEvents.evtx.**

The above allows remote log managers (including HawkEye Retriever) to be able to access the **evtx file** via the DCE/RPC mechanism in Windows.

Example of Advanced Properties File

```
#######################################################
# Sensage Windows Event Retriever Agentless Properties
#######################################################

# Using the Simplified Syntax to configure the Windows Retriever:
# There are two ways to use the simplified syntax.
# If you wish to specify a single set of pipeline values that apply to all readers follow the
example in the SINGLE_PIPELINE_TYPE section
# If you wish to specify multiple types of pipelines so that different sets of readers can use
different pipeline
# values and write to different writers, then follow the example in the MULTIPLE_PIPELINE_TYPES
section
# The information in the COMMON section applies to all configurations using the Simplified Syntax

#######################################################
# SINGLE_PIPELINE_TYPE <Begin>
#######################################################
# Specify at least one reader.
# Multiple readers should be numbered (i.e. reader1, reader2, reader3, etc.)
# Reader definitions can be of the following form:
# reader1=windows,<hostIPaddr>,<user>,<password>[,<sec:sys:app:dns:frs:dir>]
#  The last argument is optional and specifies the log types to read.
#  sec=Security, sys=System, app=Application, dns=DNS Server, frs=File Replication Service,
dir=Directory Service
#  If omitted, all log types are read. One or more may be specified in any combination
#  separated by colons. E.g. "sec:app" will read Security and Application events.
#  <user> can be of the form:
#   username
#   username@domain
#   domain\\username
#  NOTE:If the password is hashed and has been removed and replaced with *****  then any
modifications
#   to the reader will require erasing the ***** and replacing that with the appropriate password
# reader2=file,<filename>
# reader3=tcp,<listenPort>

#reader1=

# Specify exactly one or more writers.
# Writer definition can be one of the following forms:
# writer=collector,<rolloverThresholdMinutes>,<rolloverThresholdBytes>
# writer=tcp,<hostIPaddr>,<port>
# writer=syslog,<syslogHost>,<syslogPort>
# writer=file,<filename>
#writer=

# Pipeline settings: <delay in milliseconds>, <requestcount>, <zerocounter>
# Example:
#   Every second
#   Read one buffer worth of data
#   If after 60 reads there is no data, check to see if the server was reset
pipelines=1000,1,60

#######################################################
# SINGLE_PIPELINE_TYPE <End>
#######################################################

#######################################################
# MULTIPLE_PIPELINE_TYPES <Begin>
#######################################################
# Specify at least one reader.
# Multiple readers should be numbered (i.e. reader1, reader2, reader3, etc.)
# Reader definitions can be of the following form:
# reader1=windows,<hostIPaddr>,<user>,<password>[,<sec:sys:app:dns:frs:dir>],<pipeline_name>
#  The second to last argument is optional and specifies the log types to read.
#  sec=Security, sys=System, app=Application, dns=DNS Server, frs=File Replication Service,
dir=Directory Service
#  If omitted, all log types are read. One or more may be specified in any combination
#  separated by colons. E.g. "sec:app" will read Security and Application events.
#  <user> can be of the form:
#       username
#       username@domain
#       domain\\username
#  If the password is hashed and has been removed and replaced with *****  then any modifications
#       to the reader will require erasing the ***** and replacing that with the appropriate
password
# reader1=file,<filename>,<pipeline_name>
# reader1=tcp,<listenPort>,<pipeline_name>
# NOTE: <pipeline_name> should be one of the named pipeline types defined below
# Example: Three readers, two servicing high volume servers and one servicing a low volume server
# reader1=windows,1.2.3.4,Administrator,password,sec:sys,pipeline_highvolume
# reader2=windows,1.2.3.5,Administrator,password,sec:sys,pipeline_highvolume
# reader3=windows,1.2.3.6,Administrator,password,sec:sys,pipeline_lowvolume

# Specify one or more writers
# Multiple writers should be numbered (i.e. writer1, writer2, writer3, etc.)
# Writer definition can be one of the following forms:
# writer=collector,<rolloverThresholdMinutes>,<rolloverThresholdBytes>[,filename_prefix]
#   filename_prefix is a string that will be prepended to the filenames created by this writer
# writer1=tcp,<hostIPaddr>,<port>
# writer1=syslog,<syslogHost>,<syslogPort>
# writer1=file,<filename>
# Example: Three writers, one tcp writer and two collector writers where one is creating
# files that begin with 'win2k8'
#writer1=tcp,10.0.0.1,1025
#writer2=collector,60,50000000
#writer3=collector,60,50000000,win2k8

# pipelines types: you must define one of more pipeline types.
# each pipeline should be named pipeline<unique_name> where <unique_name> can be a number or
# identifying string (for example pipeline_highvolume, pipeline_lowvolume, pipeline_windows2008,
etc)
# Pipeline settings: <delay in milliseconds>, <requestcount>, <zerocounter>,<first
writer>[,<second writer>]
# Example:
# two pipelines, one for high volume data and one for low volume servers
#   Every second, Read one buffer worth of data, if after 60 reads there is no data, check to see
if the server was reset
#pipeline_highvolume=1000,1,60,writer1,writer3
#pipeline_lowvolume=1000,1,60,writer2
#######################################################
# MULTIPLE_PIPELINE_TYPES <End>
#######################################################

#######################################################
# COMMON <Begin>
#######################################################

#
# Remote Shell setting: <listenPort>, [localhostOnly]
# The second argument "localhostOnly" is optional and
# specifies that only connections made from the localHost will be accepted.
remoteshell=9999,localhostOnly

# Log level and output file.
# Valid log levels are: TRACE, DEBUG, INFO, WARN, ERROR, FATAL
```

```
logging=INFO,sample log name/windowsretriever.log

# Global State Database
# (Note incorrect use of this option can cause serious problems for the windows retriever
#  it should only be used in cases where you are defining a second retriever instance with a unique
#  agenltess.prop file)
# Sets the name (you can also specify a unique path, otherwise it defaults to
#  the path of the collector state files) of the state database.
# for example global.state.db.file=winret_2008
# global.state.db.file=

# Start From Latest
# uncommenting this option will cause new pipelines to start reading from the most recent record,
# instead of the oldest record.  This feature is useful if you are adding new readers and do not
# a slow connection to get flooded with old data or if you lost your state file info and don't want
# to reload a lot of historical data
#event.reader.startfromlatest=true;

#######################################################
# COMMON <End>
#######################################################
# FOR EXAMPLE OF CONFIGURING DISCOVERS or using the Advanced Syntax, SEE THE DOCUMENTATION
```

MANAGING REMOTE AGENTS

The HawkEye Retriever provides for the central configuration and management of multiple remote agents. When an agent is remotely managed it only requires a small amount of generic configuration information; once initialized, the agent can be run from a centralized host. Normally remote management is used in cases where a large number of remote agents exist that have similar configurations (for example, a collection of hosts that all need agents to read the same IIS log files), or in cases where there is no easy access to manage remote agents.

Configuring the Remote Agent

To associate the remote agent with a remote manager, the remote agent must be configured with the following properties in the .prop file:

1. Specify whether the remote agent (host) is to attempt contact with a remote manager; set the value to "true" as shown below:

```
event.remoteagent.enabled=true
```

2. Provide a comma-separated list of host names that are remote managers possessing configuration data for this remote agent (host).

```
event.remoteagent.hosts=<host list>
```

3. Specify the port that will be used to contact the remote manager:

```
event.remoteagent.port=<port-number>
```

Each remote agent is installed with a simple properties (.prop) file used only to establish connection configuration to obtain its own configuration from the remote manager:

- When the remote agent is started (either manually or when it is started as a service), the remote agent uses the configuration information in the properties file to contact the central host (remote manager) which in turn provides the remote agent with the actual configuration setting it requires.
- After that connection is established, the remote agent listens for management requests from the central host (remote manager) and optionally, provides path information required to ship its logs to the remote manager.
- The data the remote agent is permitted to read and where that data is written is controlled by the configuration the remote agent receives. A remote agent could get a configuration, for example, from **hostA** that specifies it writes data using a tcp writer to **hostB**.

Configuring the Central Remote Manager

You can designate one or more HawkEye Retriever instances to act as remote managers. A designated HawkEye Retriever instance that is a remote manager also has the capability to run, optionally, on the collector. (Note that if this arrangement is used, it has no affect on the HawkEye Retriever instance’s role as remote manager of other remote agents). Furthermore, a HawkEye Retriever instance can still be used to define normal readers and writers since remote management does not typically require extra overhead.

To configure the remote manager to start managing remote agents, provide the following values in its properties (.prop) file:

1. Specify the port that this remote manager will listen on for management requests:

```
event.remotemanager.port=<port-number>
```

2. Specify whether the remote manager (host) is to attempt contact with other remote agents; set the value to "true" as shown below:

```
event.remotemanager.enabled=true
```

3. Specify the path where logs shipped from remote agents will be written to the remote manager. The path should be different then the path of the log file specified for the remote manager itself.

```
event.remotemanager.logdest=<fully-qualified log file name>
```

Defining a Remote Configuration

In addition, the remote manager defines one or more remote configurations. A remote configuration might apply to a single host or multiple hosts. The goal is to keep the number of configurations as small as possible while still addressing the varying needs of your remote agents.

Here are the properties used to define a remote configuration in the remote manager's .prop file:

1. Provide a comma-separated list of one or more hosts that this remote configuration applies to. (If not supplied then any host (remote agent) requesting this configuration name will be allowed to retrieve the configuration data):

```
event.remotemanager.configs.<configuration name>.hostnames=<string>
```

2. Provide a reference to a string in the `commonconfigmap`.Typically, your `commonconfigmap` entry will have configuration data that is common to a large number of remote configurations:

```
event.remotemanager.configs.<configuration name>.commonconfig=<named common
config>
```

3. Provide a string of text that represents the properties that will be sent to the remote agent (combined with anything specified in the `commonconfigmap` entry). Separate lines by a **\n** character.

```
event.remotemanager.configs.<configuration name>.config=<string>
```

4. If remote logging is specified, provide the port number that the remote agent will use to ship logs to this remote manager.

NOTE: You specify remote logging by using the keyword **REMOTE** instead of a path name in your logging specification.

```
event.remotemanager.configs.<configuration name>.logport=<port-number for
logging>
```

AN EXAMPLE OF A REMOTE MANAGER AGENTLESS.PROP FILE

In this example, the following agentless.prop file includes two remote configurations:

- hosts that collect the following: **iis** log data and windows event data (shipping it back to the centralized collector on port 5755)
- remote configuration for hosts that read the SQL Server 2000 audit log data (shipping it back to this remote manager on port 5756.)

Both hosts use the same block of `commonconfig` data that specifies **REMOTE** in the logging setting (implying that these remotes will ship all logs back to this remote manager).

```
event.remotemanager.port=9800
event.remotemanager.enabled=true
event.remotemanager.logdest=/opt/sensage/var/log/collector/remoteagents.log
event.remotemanager.configs.iisremotes.hostnames=hosta,hostb

event.remotemanager.configs.iisremotes.commonconfig=basicremoteconfig
event.remotemanager.configs.iisremotes.config=”reader1=file,c:\\iislogs\\iis*.lo
g\nreader2=windows,localhost,Administrator,password,sec\n
writer1=tcp,collector1,5755”

event.remotemanager.configs.iisremotes.logport=9900
event.remotemanager.configs. sqlserver2000remotes.hostnames=hostc,hostd
event.remotemanager.configs.sqlserver2000remotes.commonconfig=basicremoteconfig

event.remotemanager.configs.
sqlserver2000remotes.config=”reader1=sqlserver2000auditlogreader,localohst,admin
,password,1433,c:\\sensage\\SqlServerFull.trace,,,2\n
writer1=tcp,collector1,5756”

event.remotemanager.configs. sqlserver2000remotes.logport=9900
commonconfigmap.basicremoteconfig="pipelines=1000,1,60\nlogging=ERROR,REMOTE\nre
moteshell=9900,localhostOnly"
```

Managing Remote Agents with Windows Retriever

The windowsretrievershell allows you to remotely manage any agents being managed by an instance of the HawkEye Retriever. The command is:

```
windowsretreivershell localhost <port-number> manageremotes <hostname|remote
configuration name|ALLREMOTES> <command> <command options>
```

The first parameter to the manageremotes command is one of the following:

- ***hostname***—sends the command only to that host.
- ***remote configuration***—sends the command to any hosts currently being managed with that configuration.
- **ALLREMOTES**— sends the command to any hosts currently being managed by this host.

After specifying the first parameter you then supply whatever shell command you want to execute (plus any parameters that it requires).

Example 1:

To get `stateall` output from a remotely managed host named **hostA,** run the following:

```
windowsretreivershell localhost 9999 manageremotes hostA stateall
```

NOTE: Port 9999 is the remote management port on the host you are running from,; it is NOT the remote management port of the host you are trying to access.

Example 2:

To stop the pipelines on all hosts associated with a remote configuration named **myremoteconfig**, run

```
windowsretreivershell localhost 9999 manageremotes myremoteconfig stopall
```

Example 3:

To tell every host (remote agent) that is managed by this remote manager to reload its properties file (which would force the remote agent to pick up any changes you have made to the configurations), run:

windowsretreivershell localhost 9999 manageremotes ALLREMOTES reloadpropfile CONFIRM

USING DATABASE RETRIEVERS

Using the retriever to collect data from various databases will require some considerations beyond basic retriever setup. There are some general configuration settings that are common to either type of database access, discussed next, and there are specific instructions for each type.

- "Using Database Retrievers", next
- "Oracle Database Retrievers (BETA)", on page 115
- "Password Managers for Oracle Retrievers (BETA)", on page 115
- "SQL Server Database Retrievers", on page 116
- "Defining a Metadata Function", on page 117
- "Performance Considerations for SQL Server Audit Data", on page 120
- "Maximizing SQL Server Audit Log Reader Performance (sqlserverauditlogreader)", on page 121

General DB Retriever Considerations

All database retrievers follow the same basic mode of operation once the pipeline is established:

- Every X seconds (as defined by the pipeline polling rate for that reader) a JDBC connection is reused to make a SQL query against the database.
- The data returned must include some extended timestamp column which the EDW can use to load the data and to track the last record read for subsequent reads.

Database retrievers will in most cases pull all records that satisfy a given query, as such the polling interval is best defined so that each SQL query to the database returns a reasonable amount of data. For example a short polling interval on a quiet system would result in lots of reads of zero records, while too long a polling interval on a very busy server could result in the retriever falling behind.

Data from the record set is sent to the writer or writers for that reader. At this point the data is just delimited lines of text that the Collector or other writers will process appropriately.

NOTE: The zero counter in the pipelines settings is used by database retriever (SQL Retriever) to check the trace status on the SQL Server and if the trace has stopped, an attempt is made to start it.

The Oracle Database Retriever and the generic SQL Server table reader both use the SQL String Map. The map allows you to create named pieces of SQL that can then be used in one or more readers.

You create SQL String Map Entries like this:

```
 sqlstringmap.<uniquename>="select <columns> from <table> where
[EXTENDEDTSCOL];"
```

You can create as complex a SQL query as you desire as long as the where clause includes `[EXTENDEDTSCOL]`. This marker is used by the EDW to know where to insert SQL to restrict the record set. For example the map entry:

```
sqlstringmap.oracle10i="select * from dba_audit_trail where [EXTENDEDTSCOL] AND
username='Scott';"
```

would further restrict the recordset to rows where the username was equal to the string `'Scott'`

Like all readers, database reads must still define a valid writer and pipeline type. For example a minimum `agentles.prop` to read SQL Server data from a table called **tableA** would look like this:

```
readersqltbla=sqlservertable,10.0.1.15,bob,bobpassword,1433,tableAsql,mytscolumn
,tableA,2,sqlserver_pipeline
sqlstringmap.tableAsql=select * from tableA where userid = 'sally'
writer=collector,50,50000000
pipelines=10000,1,60
remoteshell=9999,localhostOnly
logging=INFO,../logs/winretriever.log
```

This would again represent a minimal configuration.   More complex environments might include other readers, multiple pipeline types, multiple sqlstringmap entries, password managers, etc.

Oracle Database Retrievers (BETA)

Oracle Audit Logs are just Oracle tables so the same reader can be used to read either kind of data. The important element in defining an Oracle reader is the sqlstringmap property. This value references a named SQL String in the SQL String Map. Multiple readers can reference the same SQL String, but a configuration might include several named SQL strings for different kinds of auditing. As an example you might define a SQL String Map entry for reading regular Oracle audit logs on some hosts, and Fine Grained Audit Logs on other hosts.

As with any data source you must make sure that the data you pull back from an Oracle server matches the expected format defined in the PTL. Also bringing back fewer columns (by including more columns in your SELECT) or fewer rows (by restricting data in your where clause) that would otherwise not pass the PTL will greatly increase the performance and the impact of the retriever on the database.

Password Managers for Oracle Retrievers (BETA)

For Oracle Retrievers you can optionally define a password manager. The purpose of a password manager is to periodically reset the password for the Oracle user and then track that new password in the secure keystore on the retriever. You define a password manager with the following properties:

- **maxage**—An integer that indicates how old (in days) a password can be before it should be changed.
- **depth**—An integer specifying how many historical passwords to retain to prevent using the same password.

- **minpasswordlength**—The minimum length of a generated password.
- **maxpasswordlength**—The maximum length of a generated password.
- **allowablecharacters**—A string that can contain any characters allowable for use in a password on that host. Otherwise only A-Z, a-z and 0-9 are used.

For Example here is the definition of the password manager called **mypm**:

```
event.passwordmanager.mypm.maxage=30
event.passwordmanager.mypm.depth=5
event.passwordmanager.mypm.minpasswordlength=8
event.passwordmanager.mypm.maxpasswordlength=10
event.passwordmanager.mypm.allowablecharacters=$%&
```

This would create a password manager that tries every 30 days to generate a password between 8 and 10 characters. The password can include the **$, %**, and **&** characters in addition to any combination of letters and numbers. The retriever will remember the last 5 passwords it created to prevent duplicates.

There is one extra property that affects all password managers:

```
event.passwordmanager.pollinginterval=<number of days>
```

This pollinginterval specifies how often the password manager should wake up to check passwords. The less frequently the password manager runs the less system impact it will have.

SQL Server Database Retrievers

There are three types of SQL Server retrievers:

- The sqlservertable reader is used to read non-trace data contained in a table. If you are pulling non-trace data then you must have a PTL that will properly load whatever data you are querying. You can specify a namedsqlstring that is as complex as possible in the limits of TSQL which could pull back data from multiple tables, views, etc. The only restriction is that the query must be able to query time ordered data such that subsequent queries don't keep pulling back duplicate records.
- The sqlserverauditlog reader is used to collect data generated by a trace. In defining the reader you must supply a file local to the retriever that contains a stored procedure to create the trace.

  Every time the pipeline starts (or when an error has occurred reading from the pipeline) the retriever will check if there is a running trace whose output file matches that defined in your locally defined trace file.

  If the trace is not currently running, the retriever will attempt to start it, but if that fails it will attempt to execute the stored procedure defined in your trace file and start the trace from scratch

  Once the trace is running the retriever will, at every polling cycle as defined by the pipeline, connect to the SQL Server and query back any new records in the virtual table that SQL Server provides for a running trace.
- The sqlserver2000auditlogreader is used to collect data generated by a trace. In defining the reader you must supply a file local to the retriever that contains a stored procedure to create the trace. To function correctly this reader must be run on the SQL Server host as part of an HawkEye Retriever agent installations.

  Every time the pipeline starts (or when an error has occurred reading from the pipeline) the retriever will check if there is a running trace whose output file matches that defined in your locally defined trace file.

If the trace is not currently running, the retriever will attempt to start it, but if that fails it will attempt to execute the stored procedure defined in your trace file and start the trace from scratch

Once the trace is running the retriever will, at every polling cycle as defined by the pipeline, connect to the SQL Server and query back any new records in the virtual table that SQL Server provides for a running trace.

At an interval specified in the `agentless.prop` file the retriever will stop the trace, attempt to remove extra trace files, and then restart the trace. There are two `agentless.prop` entries related to this functionality:

- A setting to control how many **.trc** files should be kept on the system. SQL Server numbers **.trc** files starting from **tracename.trc** to **tracename_1.trc** to **tracename_2.trc**, etc.:

```
event.reader.ss2kfilemax=<number of files to keep>
```

- A setting to control how often (in seconds) the retriever should scan for old files (the more often you scan for files the less efficient the process will be since it involves stopping and starting the trace):

```
event.reader.scrubfilesinterval=<number of seconds>
```

The trace file you define should follow one of the included examples, **SQLServer2000.trace** or **SQLServerFull.trace**. The important points are:

The `SET @traceFileLocation` line must reference a location on the SQL Server host that is writable by SQL Server. In the case of SQL Server 2000 the account that is used to run the HawkEye Retriever agent must also have write access to that directory since the retriever will need to be able to remove **.trc** files as needed (again only for SQL Server 2000) .

The **@tracefilelocation** can often be determined programmatically (depending on domain name, etc.) but any trace definition must compute the **@tracefilelocation** from a fixed variable that must be declared as **@TRACE_BASEFILENAME**. For example, regardless of how you compute **@tracefilelocation**, you must at minimum declare it like this:

```
DECLARE @traceFileLocation NVARCHAR(245)
DECLARE @TRACE_BASEFILENAME NVARCHAR(245)
SET @TRACE_BASEFILENAME = ’sqlauditfile’
SET @traceFileLocation = ’c:\trace\’ + @TRACE_BASEFILENAME
```

Use the value of **@TRACE_BASEFILENAME** to ensure the correct trace is being analyzed.

NOTE: @TRACE_BASEFILENAME should be a name that is unique from other traces on the system.

SetEvent calls that are used to turn on polling for specific events and columns must match the PTL used to collect the data..

For Sql Server 2000 trace files the **sp_trace_create** call should ***NOT*** include the **@filecount** variable. SqlServer 2000 does not have the ability to automatically clean up older **.trc** files (which is why the sqlserver2000auditlogreader must reside on the windows host itself to do the cleanup), subsequent versions will clean out older **.trc** files based on the **@filecount** option passed to **sp_trace_create.**

Defining a Metadata Function

In order to maximize SQL Server reader performance for very large traces, the reader needs to know information about the trace files that exist in the trace. To provide the reader with this information, you must define a metadata function on the server (which you then reference in the reader definition). Refer to the example function and reader definition below.

```
<code formatting.>
/*

This is an example of the function used to track trace file metadata for a SQL
Server Audit Log reader.

If you specify a function name in your reader definition then a function must
exist on the server similar to this example.

The function takes two argument:

The path to where the trace file reside
The name of the trace.

The function must return a table with 4 columns: fileName, size, lastModified,
and error

Each row should have either a non-null error value or a filename, file size in
bytes, and a date the trace file was last modified.

This information allows SQL Server to make sure it is only querying relevant
trace files and prevents the reader from consuming too much data on very large
traces.

*/
USE [master]
GO
/****** Object: UserDefinedFunction [dbo].[getTraceFiles] Script Date: 12/30/
2013 3:04:59 PM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE FUNCTION [dbo].[getTraceFiles](@folder nvarchar(260),@tracefilename
nvarchar(260))
RETURNS @TraceFileMetaData TABLE
(
[fileName] VARCHAR(260),
[size] BIGINT,
[lastModified] DATETIME,
[error] VARCHAR(500)
) AS
BEGIN
DECLARE @handler INT, @objApp INT, @objError INT,@strErrorMessage NVARCHAR(500),
@Command VARCHAR(1000),@objFolder INT,@objFolderItems INT,@moddate datetime,
@fileCount INT,
@fileName VARCHAR(260), @filePath VARCHAR(260), @fileSize bigint, @objItem INT
---------------------------------
SELECT @strErrorMessage = 'Opening Shell.Application'
EXECUTE @handler = sp_OACreate 'Shell.Application', @objApp OUT
IF @handler = 0
SELECT @objError = @objApp, @strErrorMessage = 'Opening NameSpace Object',
@command = 'NameSpace("+@folder+")'
IF @handler = 0
EXECUTE @handler = sp_OAMethod @objApp, @command,
@objFolder OUT
IF @objFolder IS NULL
BEGIN
SELECT @strErrorMessage = 'ERROR ' + @strErrorMessage
INSERT INTO @TraceFileMetaData(error) SELECT @strErrorMessage
```

```
END
ELSE
BEGIN
SELECT @objError = @objFolder,
@strErrorMessage = 'Getting count of matching files"',
@command = 'Items.Count'
END
IF @handler = 0
EXECUTE @handler = sp_OAMethod @objfolder, @command,
@fileCount OUT
IF @handler = 0
SELECT @objError = @objFolder,
@strErrorMessage = 'Getting list of matching files',
@command='items()'
IF @handler = 0
EXECUTE @handler = sp_OAMethod @objFolder,
@command, @objFolderItems OUTPUT
SELECT @fileCount = @fileCount - 1
WHILE @handler = 0 AND @fileCount >= 0
BEGIN
IF @handler = 0
SELECT @objError = @objFolderItems,
@strErrorMessage = ' getting file object',
@command='item(' + CAST(@fileCount AS VARCHAR(32))+')'
IF @handler = 0
EXECUTE @handler = sp_OAMethod @objFolderItems,
@command, @objItem OUTPUT
IF @handler = 0
SELECT @objError = @objItem,
@strErrorMessage = ' getting file properties'
IF @handler = 0
BEGIN
EXECUTE @handler = sp_OAMethod @objItem,
'path', @filePath OUTPUT
END
IF @handler = 0
BEGIN
EXECUTE @handler = sp_OAMethod @objItem,
'name', @fileName OUTPUT
IF LEFT(@fileName,LEN(@tracefilename)) != @tracefilename
BEGIN
SELECT @fileCount = @fileCount - 1
CONTINUE
END
END
IF @handler = 0
EXECUTE @handler = sp_OAMethod @objItem,
'ModifyDate', @moddate OUTPUT
IF @handler = 0
EXECUTE @handler = sp_OAMethod @objItem,
'Size', @fileSize OUTPUT
INSERT INTO @TraceFileMetaData ([fileName], size, lastModified)
SELECT @filePAth, @fileSize, @moddate
IF @handler = 0 EXECUTE sp_OADestroy @objItem
SELECT @fileCount = @fileCount - 1
END
IF @handler <> 0
BEGIN
INSERT INTO @TraceFileMetaData(error) SELECT @strErrorMessage
```

```
END
EXECUTE sp_OADestroy @objFolder
EXECUTE sp_OADestroy @objApp
RETURN
END
<end of code formatting>
```

NOTE: Among several ways, the example above provides just one way to implement the MetaData function in SQL Server. As long as the data is returned with the correct number of rows based on the inputs, you can implement this function in line with your security practices.

Performance Considerations for SQL Server Audit Data

SQL Server traces comprise *X* number of files that are each *Y* megabytes in size, where *X* and *Y* are determined by the settings in your trace definition. For example, assume these settings are in the supplied trace:

```
SET @maxfilesize = 10
SET @filecount = 10
```

These settings mean that the trace will, when full, span 10 files of 10MB each. For higher volume servers you might need upwards of 30-50GB of trace data or more kept on the system. These trace files are written sequentially so in the above example a fully-filled trace might have 10 files each of 10MB in size. Unless you have specified a Metadata function, as explained in the previous section, then each query to the system will attempt to pull the entire trace into memory on the SQL Server host even if your query only needs the most recently generated records. Because of this design, here are some general guidelines to keep your system performing with minimal impact to your SQL Server instances:

If you use a MetaData function:

The default polling interval of 1000ms is much too fast. Depending on volume and your requirements in getting into the retriever, you will want to poll as infrequently as possible while still meeting your goals.

A simple query to determine (roughly) how fast data is coming in would be:

```
select count(startTime) as recordCount, max(startTime) as maxTime,
min(startTime) as minTime from FROM
sys.fn_trace_gettable('c:\trace\mytrace.trc',1) where startTime is not null;
```

(where **'c:\trace\mytrace.trc'** is the appropriate file location on your system)

Comparing the record count with the delta between **minTime** and **maxTime** provides a rough idea of how many events per second were being generated. The size of the event will vary depending on the trace definition, but if there are > 1 **.trc** files in your trace directory, you can safely divide the size of the first file by the number of records in recordCount to get an average event size.

Once you have a sense of how fast data is coming in, simply adjust the **maxRecordPerPoll** value and the polling interval to make sure you are capturing data faster than SQL Server is generating it. For example, if you are generating 10K events per second, then you might want to poll for 50K records every 3000ms. For lower volume servers this exercise is less important than on higher volume boxes.

Be sure to strike a balance between **@maxfilesize** and **@filecount**; you do not want either value to be very large compared to the other. For example, setting **@filecount** to **10K** with **@maxfilesize** of **1MB** will create so many trace files that Windows will have trouble managing them. Conversely, setting the **@filecount** to **3** with a **@maxfilesize** of **30GB** will still cause the reader to make memory intensive queries because querying a single **.trc** file will require SQL Server to pull 30GB of data into memory.

A good rule of thumb (once you have determined how much trace data you need to keep on the machine) is to divide the values between maxfilesize and filecount up evenly until you reach maxfilesizes of **~100MB**, at which point just start increasing only the filecount to reach your target trace size.

If you cannot use/define a Metadata function:

Without metadata information any query to the trace is required to pull the whole trace into memory. This means that each query will have a significant performance impact on servers with large traces. To reduce impact:

- Make your polling interval as long as possible since this will reduce the amount of overhead to SQL Server

- Consider reducing the values for **@maxfilesize** and **@filecount** so that the trace is of a more manageable size.

NOTE: Gaps in SQL Server traces:

For various reasons SQL Server can experience a 'gap' (usually in response to system problems) in the trace files. For example, there might be a **sqlauditfile_10.trc** and a **sqlauditfile_12.trc** but **sqlauditfile _11.trc** appears to be missing. The reader will attempt to skip over any gaps it encounters but this is an error condition that should be fixed by SQL Server Administrators. The following is the normal route to fixing this error condition:

1. Stop the trace.

2. Delete all the **.trc** files for that trace name.

3. Restart the trace.

Maximizing SQL Server Audit Log Reader Performance (sqlserverauditlogreader)

By default, **sqlserverauditogreader** has no knowledge of how the trace is actually stored on disk. This can cause performance issues as the trace grows larger (over 10GB) because each query must in effect do a table scan over the entire trace even though the most recent data is most likely the one of interest.

To prevent this performance impact there is an optional flag in the reader definition referred to as **SQLServer_traceFileMetaDataFunctionName.sql**. This value should match the name of a function on SQL Server instances that take two inputs (a string that is the directory where trace files are stored and a string that is the name of the trace) and returns a table with the following four columns:

- fileName (a String that is the full path to a trace file)

- size (a bigint that is the trace file’s size in bytes)

- last Modified (a timestamp that is the last time the trace file was written to)

- error (normally a string set to null, but if errors occur it can pass back an error string to aid analysis)

SQLServer_traceFileMetaDataFunctionName.sql is included as an example of how the function could be written. The function can be called like this:

```
SELECT * FROM [(dbo].[getTraceFiles] (’c\trace’,’sqlauditfile") order by
lastmodified;
```

The following list of rows is returned for each **.trc** file that the trace has created:

fileName	Size	lastModified	Error
C:\trace\sqlauditfile_2.trc	104857600	2013-11-12 14:23:34.000	NULL
C:\trace\sqlauditfile_3.trc	104857600	2013-11-12 14:23:08.000	NULL
C:\trace\sqlauditfile_4.trc	104857600	2013-11-12 14:29:34.000	NULL
C:\trace\sqlauditfile_5.trc	104857600	2013-11-12 15:41:12.000	NULL

This function ensures that only relevant **.trc** files are searched, greatly reducing the impact of the reader.

Performance Details and Recommendations for Configuration

For SQL Server Audit Log Reader the following calculation was used to determine performance for "max" file size, for "n" files (n max and n avg) retrieved in time "t":

- Maximum file size tested: 100 MB
- Maximum number of files tested: 300 files
- The "avg" file size tested for "n" files retrieved in time "t" was performed on different file size trace files: 1MB, 10MB, 30MB & 100MB

The retriever pulling rate should be larger than the SQL Server audit log arrival rate .Otherwise retriever will fall behind and there is chance to miss trace data that gets deleted by SQL Server when files are rolled over as time passes. You can derive the recommended configuration by answering the following questions.

- How many records in a 100MB size log trace file?
- How long does it take to fill a 100MB log trace file?

For example, if there are 60,000 records in a 100MB size log trace file which gets filled in 3 minutes that means that there are 333 log events created per second on the SQL Server. So the retriever should be configured to fetch more than 333 records per polling cycle. For example, if the retriever polling cycle is 10 seconds then number of records to pull would be 3330.

CHAPTER 5

Configuring Netflow Receiver for Data Collection and Analysis

The Netflow Receiver refers to the HawkEye AP process of collecting and analyzing network data, based on Netflow processing tools such as **nfcapd** and **nfdump**. These programs are part of the NFDUMP flow-tools set that processes data in Netflow's flow-record format (supported versions include: v1, v5, v7, v9, and IPFIX), as well as a limited set of sflow. The NFDUMP fllow-tools are IPv6 compatible. Included also is support for CISCO ASA (NSEL) and CISCO NAT (NEL) devices, which export event logging records as v9 flows.

This chapter contains these sections:

How Netflow Receiver Works

Netflow provides Network Managers a detailed view of IP traffic flows on the network. Routers and Layer3 switches act as sensors pushing netflow data, with each router representing network traffic flow based on source and destination IP address, source and destination port, Layer 3 protocol type, and type of service.

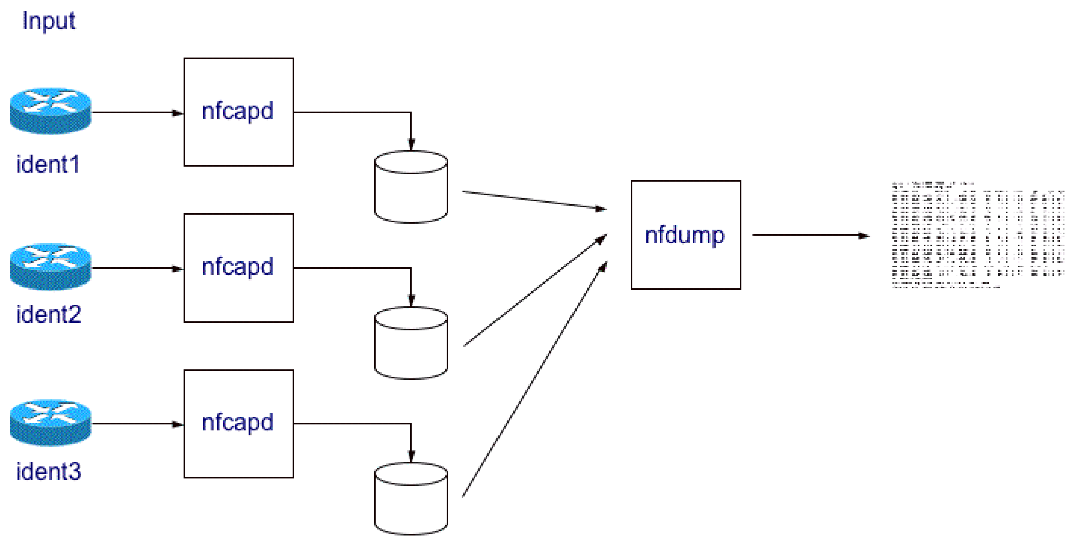
Network Receiver requires one **nfcapd** process to capture data for each netflow stream in your configuration. The Netflow client uses **nfcapd** to capture data from Netflow sources and forwards that data on to the Netflow Receiver (Netflow server processes) running on the Collector System.

Each router that will store Netflow data must be configured to export that data. The **nfcapd** program performs a 60-second file rotation, preventing excessive system load and offering reliability in the event of a network outage or a DDOS attack overload.

A Central processing node runs modified **nfdump** program(s) that pick up the incoming (new) files through a daemon process that establishes an ssh connection (shared key) to the central processing node. The new data is processed into streamed output through a program similar to the SNMP preprocessor. This program forwards the network data to the parser and also bundles that data into large files for Collector consumption.

If the connection fails, the daemon process retries until the connection is reestablished. Data remains in files and is not lost. To prevent data loss, the remote connection confirms each incoming chunk of data. The ssh program is used to transmit data in an encrypted and authenticated manner.

Figure 5-1: Netflow Receiver Configuration



When **nfdump** runs periodic queries against recently loaded data. and aggregates the result data, storing it in the EDW, the TrickleFeed Load can be enabled so that the data can be ingested faster. For details, see the **--trickle** option in Chapter 3: Loading, Querying, and Managing the EDW of the *Administration Guide.*

The streaming environment allows for real-time alerts on the data and insertion into the Collector either through sylogng or a variant of the HawkEye AP preprocess program. Data is transmitted to the TCP Port configured in the Config file.

The captured files will be stored in the directory specified in the config.file and the files can be printed with the NFDump flow-tools set.

NETFLOW RECEIVER SECURITY REQUIREMENTS

The tools comprising the Netflow Receiver do not require root privileges, unless you are using the following port: **<1024**. No access control mechanism exists in **nfcapd.** It is assumed that the Network Administrator has established host level security to filter the proper IP addresses.

CONFIGURING NETFLOW RECEIVER

To configure each data source in your Netflow Receiver configuration or specify options for data display output, you use the client and server **.prop** files, which are described in this section.

This section contains the following topics:

- “Configuring Network Sources to Export Netflow Data”, next
- “Specifying Netflow Receiver Output Formats”, on page 126

- “Printing IPv6 Records”, on page 129
- “Capturing Netflow Data (nfcapd)”, on page 129
- “Configuring Netflow Data Display and Analysis (nfdump)”, on page 130
- “Reading and Converting Flow-Tools Data (ft2nfdump)”, on page 130
- “Aggregating Flows”, on page 130
- “Specifying and Displaying Top N Statistics”, on page 133
- “Anonymizing Flows”, on page 133

Configuring Network Sources to Export Netflow Data

Configure each network data router to export Netflow data. For details, see the relevant documentation for your model and the devices that your are configuring.

Sample Configuration

The following is a commonly-used generic CISCO sample with **global configuration** commands to enable Netflow on an interface:

```
interface fastethernet 0/0
ip route-cache flow
```

In this example, the following global configuration commands are issued to tell the router to send (export) the netflow data:

```
ip flow-export
ip flow-export version 5
```

Note that the version of Netflow you are using must be specified.

Specify at what interval in minutes between 1-60 that you want to divide the long-lived flows. Typically, you will specify long-lived flows be broken into 5-minute segments, as in:

```
ip flow-cache timeout active 5
```

Note that the number of minutes chosen should be equal or less than the file rotation period, which is typically 5 minutes. The file rotation period refers to when Netflow Receiver creates a new file for incoming netflow data storage. For details, refer to “Capturing Netflow Data (nfcapd)”, on page 129.

On Catalyst 6500/7600 routers, you must make sure to enable **NDE** (Netflow Data Export) in addition to the normal Netflow export. NDE is the hardware variant of Netflow export on these routers. Here is a configuration example:

```
mls flow ip interface-full
mls flow ipv6 interface-full

mls nde sender version 5
```

Note that IPv6 NDE isn’t implemented yet, but it can still be useful to be able to look at the "live" flows with "show mls netflow ipv6".

Configuration Setting Considerations

Consider the following when determining your Netflow Receiver configuration settings:

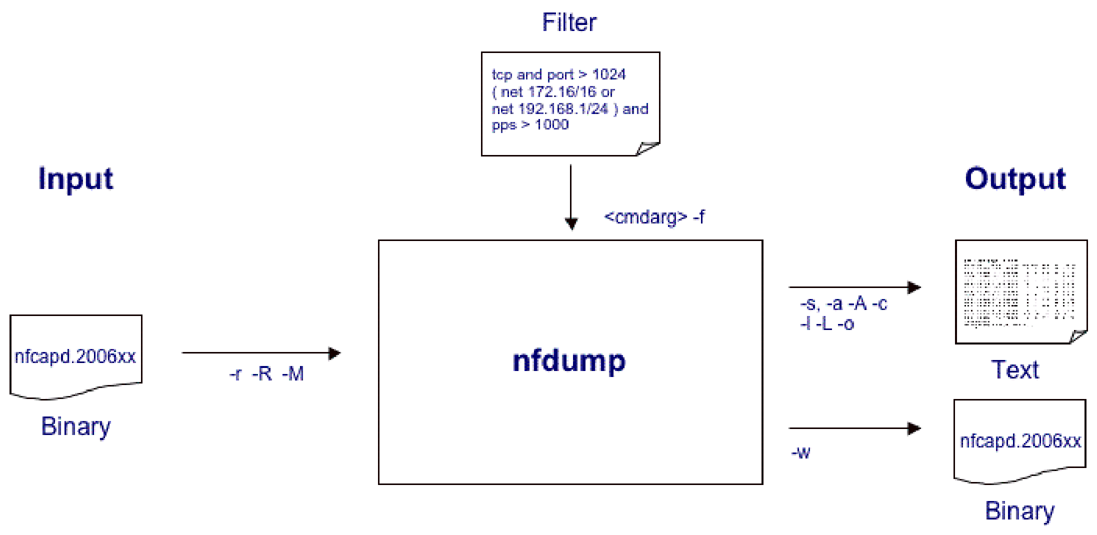
1 On a busy router, consider aggressively timing out small flows as reflected in these settings:

```
mls aging fast time 4 threshold 2
mls aging normal 32
mls aging long 900
```

In the example, you still provide the "traditional" Netflow configuration noted at the beginning of this section, which includes "ip flow ingress" or "ip route-cache flow" on every interface. This allows you to view "software-switched" flows such as those that go to the router itself. It is assumed that this is the only traffic that you are viewing.

2 Netflow versions v5 and v7 have 32-bit counter values. The number of packets or bytes may overflow this value, within the flow-cache timeout on busier routers. To prevent overflow, you may consider reducing the flow-cache timeout to lower values. All Nfdump flow-tools use 64-bit counters internally, which means all aggregated values are correctly reported.

Figure 5-2: Netflow Receiver Filtering



For more details on Netflow Receiver commands, refer to the Netflow website for a full description of each command.

Specifying Netflow Receiver Output Formats

The NFDump flow-tools has four fixed output formats: raw, line (default), long, and extended. The default format is line. You may, however, specify any desired output format using the custom format **fmt**.

The following is a description of each format type.

RAW FORMAT

The raw format displays each record in multiple lines and prints any available information in the record. The record printed is netflow version independent, but may contain different additional fields depending on the source.

This format is rarely used, but it contains all information available for the record.

Here is an example of the Flow Record in Raw format:

```
Flags       =       0x00000000

  size        =              52
  mark        =               0
  srcaddr     =    36.249.80.226
  dstaddr     =    92.98.219.116
  First       =       1125377992 [2005-08-30 06:59:52]
  Last        =       1125377992 [2005-08-30 06:59:52]
  msec_first  =             338
  msec_last   =             338
  dir         =               0
  tcp_flags   =               0
  prot        =              17
  tos         =               0
  input       =               5
  output      =               3
  srcas       =            1299
  dstas       =               0
  srcport     =            3040
  dstport     =            1434
  dPkts       =               1
  dOctets     =             404
```

LINE FORMAT

To display one line at a time, use this command:

-o line

Following is the default format with one netflow record per line specified:

```
Date flow start         Duration ProtoSec IP Addr:Port   Dat IP Addr:Port    PacketsBytes Flows
2005-08-30 06:59:52.338    0.001 UDP    36.249.80.226:3040  -> 92.98.219.116:1434    1    404     1
```

The date and duration of the flow are provided in millisecond resolution.The number of flows is always 1 unless flows are aggregated. For details on Aggregation, see "Aggregating Flows", on page 130.

LONG FORMAT

The long format displays additional information such as TCP flags, Type of Service (Tos), etc.

To display the long format on each line, use this command:

```
-o long
```

Following is a long format example with the additional information displayed:

```
Date flow start         Duration Proto Sec IP Addr:Port       Dat IP Addr:Port     Packets Bytes Flows
2005-08-30 06:59:52.338   0.001 UDP        36.249.80.226:3040 -> 92.98.219.116:1434 .AP.SF     1        404    1
```

EXTENDED FORMAT

The extended format contains format in addition to the long format which includes the following information: pps (packet per second), bps (bits per second), and bps (bytes per packets). This additional information is calculated and displayed for each record.This format provides the most information on each line.

To display the extended format, use this command:

```
-o extended
```

Following is an extended format example with the additional information displayed and a time window of `Aug 30 2005 06:53:53 - Aug 30 2005 06:54:56`.

ADD THIS IN

CUSTOM OUTPUT FORMAT

The custom output format is the most flexible format since it specifies how the output looks. The output format is defined using element tags as well as plain ASCII text. The following is a table of predefined element tags:

Tag	Description	Tag	Description
%ts	Start time - first seen	%in	Input Interface num
%te	End time - last seen	%out	Output Interface num
%td	Duration	%pkt	Packets
%pr	Protocol	%byt	Bytes
%sa	Source Address	%fl	Flows
%da	Destination Address	%pkt	Packets
%sap	Source Address:Port	%flg	TCP Flags
%dap	Destination Address:Port	%tos	Tos
%sp	Source Port	%bps	bps - bits per second
%dp	Destination Port	%pps	pps - packets per second
%sas	Source AS	%bpp	bps - bytes per package
%das	Destination AS		

For example, to customize the -o long format you can enter the following predefined elements on the command line:

```
-o "fmt:%ts %td %pr %sap -> %dap %pkt %byt %fl"
```

Printing IPv6 Records

IPv6 addresses require more space for printing than IPv4 addresses. In order to keep the output clearly arranged, IPv6 addresses are compacted in normal output as shown below:

```
Data flow start           Duration Proto   Src IP Addr:Port         Dst IP Addr:Port         Packets
2006-03-09 11:55:03.900      0.000 ICMP6   2005:62..2c:9c10.0->     2005:620:0:c000::d.0           1

Bytes       Flows
 104           1
```

The middle part of each IPv6 address will be truncated, but should allow enough of the address to identify the record. If the full length of the IPv6 address is required, add the digit "6" to the output format. Examples are:

```
-o line6
-o long6
-o extended6
```

or you may add the option **-6** as in:

```
-o extended -6
```

Here is the output with the full address displayed:

```
Data flow start           Duration ProtoSrc IP Addr:Port                     Dst IP Addr:Port         Packets
2006-03-09 11:55:03.900      0.000 ICMP62005:620:0:8:203:baff:fe2c:9c10.0-> 2005:620:0:c000::d.0            1

(Continued)
Bytes    Flows
 104       1
```

Capturing Netflow Data (nfcapd)

To enable Netflow Receiver to capture netflow data, set up **nfcapd** (netflow capture daemon) to read netflow data from the network and store the data into files. Specify the ports that **nfcapd** listens on to capture the netflow data:

nfcapd stores all data to disk allowing for a storage process that is separate from analysis The data is organized in a time-based manner. Every *n* minutes (typically set for 5 minutes), **nfcapd** rotates and renames the output file with the following timestamp interval format: nfcapd.*YYYYMMddhhmm*; for example, the timestamp value **nfcapd.200407110845** contains data from July 11, 2004 at 08.45 onward. Based on a 5-minute interval, this results in 288 files per day.

You can specify analysis for this data on a single file, or by concatenating several files for a single run. The output is either ASCII text or binary data, which when saved into a file, is ready again for processing with the NFDump flow-tools set.

In addition, several netflow data sources are specified from locations such as **router1**, **router2** and so on as in the following sample organizational format:

```
/flow_base_dir/router1
/flow_base_dir/router2
```

In the above example, **router1** and **router 2** are sub-directories of the **flow_base_dir.**

For each of the netflow sources, you start an nfcapd process as in the examples below:

```
/nfcapd -w -D -l/flow_base_dir/router1 -p 23456
/nfcapd -w -D -l/flow_base_dir/router2 -p 23457
```

Configuring Netflow Data Display and Analysis (nfdump)

Nfdump (Netflow dump), part of the NFDump flow-tools set, provides the display and analysis of netflow data. It reads the data for display from files stored by **nfcapd** and creates lots of top N statistics of flows, IP address, ports, etc., all in the order that you specify. For details on specifying the order, see “Specifying and Displaying Top N Statistics”, on page 133.

Both **nfdump** and **nfcapd** work in conjunction with **ft2nfdump**, described below, to allow for storage and analysis of netflow data.

Reading and Converting Flow-Tools Data (ft2nfdump)

The **ft2nfdump** program is a flow-tools converter for reading, converting and storing netflow data, created with the flow-tools package. It works either as a pipe filter or reads flow-tools data format directly from a specified file or from stdin in a chain of flow-tools commands, converting the data into **nfdump** format for NFDump flow-tools processing.

The **ft2nfdump** program lets you analyze netflow data from the past as well as to track specified traffic patterns continuously. You can analyze netflow data back to a specified length of time in the past, limited only by the disk space that holds your data. The tools, which are optimized for efficient filtering speed, use syntax with filter rules that are pcap-like (similar to the widely-used tcpdump).

Aggregating Flows

You can aggregate flows by specifying the **-a** option on the command line. By default, flows that have an identical protocol, source, and destination IP address as well as identical source an destination ports are aggregated. You can change the default behavior by specifying a different aggregation scheme using the following syntax:

[**-A** <*scheme*>]

where *scheme* can be any combination of **scrip** (source IP address), **dstip** (destination IP address), **srcport** (source port), **dstport** (destination port) in a comma-separated list.

Examples:

The following shows a sample of the default report in which 10 flows have been aggregated as indicated in the last column of the output.

```
Data flow start          Duration Proto   Src IP Addr:Port         Dst IP Addr:Port         Packets
2005-03-30 06:59:54.324     250.498.TCP   63.183.112.97.9050->     146.69.72.180:51899          12

(Continued)
Bytes        Flows
2198            10
```

The following example aggregates flows with the same source IP address and destination address using the **-A** option:

```
-a -A scrip,dstport
```

The output shows flows aggregated by source IP address and destination. The number of flows that were aggregated for a given source IP address and destination are indicated in the last column of the output.

```
Date flow start          Duration Proto Src IP Addr:Port Dst IP Addr:Port Flags Tos  Packets    Bytes   pps
2005-08-30 06:59:25.137  213.697 TCP     32.249.32.48:0 -> 0.0.0.0:135   ....... 0       23     1104       0
2005-08-30 06:59:24.563  330.110 TCP   49.112.228.156:0 -> 0.0.0.0:1433  ......  0    47943     2.2M     145
2005-08-30 06:59:54.322  201.857 TCP  148.190.164.126:0 -> 0.0.0.0:36129 ......  0       10      460       0
2005-08-30 06:59:54.257   48.768 TCP      92.90.57.46:0 -> 0.0.0.0:59501 ......  0        5      230       0

(continued)
bps    Bpp Flows
4‘     48      13
55769  48   27864
18     46       6
37     46       2
```

NOTE: All other elements, not aggregated, are set to 0.

Aggregating Flows on a Subnet Level

You can also aggregate flows on a subnet level. To create appropriate masks for aggregation, you must specify the protocol version with the address field, as in:

```
-a -A scrip4/24, dstport
```

The above example aggregates flows on a **/24 IPv4** base and destination port.

Filtering Netflow Data for Display and Analysis

The NFDump flows-tools set contains a powerful and fast filter engine. All flows are filtered before they are processed through a program known as **nfprofile**. This program reads the netflow data from the files stored by **nfcapd** and filters the Netflow data according to the filter sets (profiles) you specify. **nfprofile** stores the filtered data into files for later use. If no filter is provided, all flows are processed.

A filter is specified as the last argument enclosed in single quotes (**’**)or in a specified file using the -r option.

Any line in the file starting with a pound sign (#) is treated as a comment; filter syntax is similar to the widely-used **tcpdump** syntax.

As shown in this template sample file, any filter consists of one or more expression (`expr`) and you can link any number of *expr* together:

```
       expr and expr, expr or expr, not expr,  ( expr ).
       expr can be one of the following filter primitives:
       protocol version
           inet or ipv4 for IPv4 and inet6 or ipv6  for IPv6 flows only
       protocol
           TCP, UDP, ICMP, GRE, ESP, AH, RSVP or PROTO <num> where num is the
protocol number.
       IP address
           [SourceDestination] IP a.b.c.d or
         [SourceDestination] HOST a.b.c.d with a.b.c.d as any valid IP address.
SourceDestination may be omitted.
       SourceDestination
           defines  the IP address to be selected and can be SRC, DST or any
combination of SRC and|or DST. Omitting SourceDestination is equivalent
```

```
                to SRC or DST.
            network
                [SourceDestination] NET a.b.c.d m.n.r.s
               [SourceDestination] NET a.b.c.d / num with a.b.c.d as network number,
m.n.r.s as netmask or num as maskbits respectively.   The  network
              may be given as a.b, a.b.c, where a B or C-class equivalent netmask is
assumed.
            Port
              [SourceDestination]  PORT [comp] num with num as a valid port number.
If comp is omitted, '=' is assumed.
            Interface
                [inout]  IF num with num as an interface number.
            inout
                defines the interface to be selected and can be IN or OUT.
            Flags
                flags tcpflags with tcpflags as a combination of:
                A   ACK.
                S    SYN.
                F    FIN.
               R   Reset.
                P   Push.
                U   Urgent.
                X   All flags on.
```

Note that the ordering of the flags is not relevant. Flags not mentioned are ignored. In this continuation of the sample filter template the syntax to retrieve flows with only the tcpflag **SYN** (flag option **S**) is noted below:

```
flag set, use the syntax 'flags S and not flags AFRPU'.
TOS Type of service: tos value with value 0..255.
Packets
    packets [comp] num [scale] to specify the packet count in the netflow record.
Bytes
    bytes [comp] num [scale] to specify the byte count in the netflow record.
Packets per second: Calculated value.
    pps [comp] num [scale] to specify the pps of the flow.
Duration: Calculated value
    duration [comp] num to specify the duration in milliseconds of the flow.
Bits per second: Calculated value.
    bps [comp] num [scale] to specify the bps of the flow.
Bytes per packet: Calculated value.
    bpp [comp] num [scale] to specify the bpp of the flow.
AS  [SourceDestination]  AS num with num as a valid AS number.
scale Scaling factor. Maybe k m g. Factor is 1024
comp The following comparators are supported:
=, ==, >, <, EQ, LT, GT .  If comp is omitted, '=' is assumed.
```

Examples:

The following are filter syntax examples:

```
nfdump -r /and/dir/nfcapd.200407110845 -c 100 'tcp and (src ip 172.16.17.18 or dst ip 172.16.17.19) '
nfdump -r /and/dir/nfcapd.200407110845 -A srcip,dstport 'in if 5 and net 10.0.0.0/24 and not host
10.0.0.1 and bps > 10k and duration <100 and dst port 1433'
```

Specifying and Displaying Top N Statistics

The **nfdump** programs provides a number of statistics, which you can request by supplying one or more **-s** arguments in the following syntax:

[**-n** <*num*>] **-s** *type*[**/orderby** <*order*>]

whrere:

num is the number of lines displayed in the output. Top N defaults to 10 unless **-n** <*num*> is specified. **-n 0** means an unlimited number, unless the **-s record** is specified as <*type*>, which limits <*num*> to 1000.

type can be one or more of the following for the same run:

Valid Types	Description
`record`	Statistic about aggregated netflow records Note: This type can be formatted according to the available output formats specified with the -o option. For details, see ????
`srcip`	Statistic about source IP addresses
`dstip`	Statistic about destination IP addresses
`ip`	Statistic about any (source or destination) IP addresses
`srcport`	Statistic about source ports
`dstport`	Statistic about destination ports
`port`	Statistic about any (source or destination) ports
`srcas`	Statistic about source AS numbers
`dstas`	Statistic about destination AS numbers
`as`	Statistic about any (source or destination) AS numbers
`inif`	Statistic about input interface numbers
`outif`	Statistic about output interface numbers
`if`	Statistic about any (input or output) interface numbers
`proto`	Statistic about protocol numbers

<*order*> specifies the order that the statistics is displayed, which can be by **flows**, **packets**, **bytes**, **pps**, **bps**, or **bpp**. You may specify more than one orderby, which results in the same statistic, but ordered differently. If you do not specify orderby (which is optional), statistics by default are ordered by **flows**.

Anonymizing Flows

You can make IP addresses in flows anonymous by specifying the following option:

-K <*key*>

where:

<*key*> is either a 32-character string or a 64-digit hexidecimal string starting with **0x**.

IP addresses are anonymized before they are printed or saved to file. This means that the filter applies to the original IP address.

NOTE: nfdump uses the Crypto-PAn module to anonymize IP addresses. For more details on Crypto-PAn, refer to the following website: http://www.cc.gatech.edu/computing/Telecomm/cryptopan

Converting Flow-Tools Netflow Data

The Flow-tools Converter reads flow-tools data either from stdin or from a given file specified with the **-r** option. It converts the data into nfdump format and writes nfdump format to stdout. Use the following syntax to convert a file:

ft2nfdump -r<*flow-tools-file*> | **nfdump -w** <*nfdump-file*>

You can supply any other **nfdump** command line switches to directly process flow-tools data with **nfdump**.

Cleaning up old Netflow Data (nfclean.pl)

A sample script is available to clean up old Netflow data. You may specify that this script run every hour or so.

Investigating Netflow Receiver Failures

If you need to troubleshoot a Netflow Receiver failure, you may instead want to specify logging at a high verbosity level to track down issues. You will then see log errors such as: permission denied while accessing files, error output from ssh and the like, and details of files received and transferred.

```
mkdir /tmp/mylogs
```

Edit the **.prop** file and set:

```
logging.path=/tmp/mylogs
logging.level=DEBUG
```

Restart the client or server; detailed logging is generated to help track down any issues.

Sample Netflow Receiver Configuration Files

This section contains sample configuration files for the Netflow client and server. Refer to "Configuring Netflow Receiver", on page 124.

Sample netflow_client.prop Configuration File

```
# Logging directory, if specified log files are rotated once per day, with
#'today.log' containing current log data.
# Otherwise file logging is disabled:
#logging.path=/tmp/netflow_client_logs
```

```
logging.path=

# What is the least priority message we log?
#  DEBUG
#  INFO
#  WARNING
#  ERROR
logging.level=INFO

# 1 - enable logging to syslog, 0 - disable:
logging.syslog=1

# our pid/lock file
run.pid=/tmp/netflow_client.pid

# IMPORTANT!
# This value must be unique for each netflow client!
# this is appended to file names to prevent collisions. It must also
# be a simple legal file name component. If this is not unique data
# loss will result.
netflow.clientId=example1

# This value is the econds between file rotation. For real time use you probably
# want to leave this at 1 or a low number of seconds
# This will be the nfcapd -t option
# If your nfcapd is unpatched, this will be automatically adjusted to 60 seconds
# minimum.
netflow.rotate=1

# netflow spool directory
# This will be the nfcapd -l option
netflow.spool=/tmp/netflow_client_spool

# nfcapd listen port number
#  This will be the nfcapd -p option
netflow.port=8888

# IMPORTANT NOTE:
#  Netflow data is often sent via UDP which is lossy, especially over
#  WAN connections. It is strongly recommended that the netflow_client
#  software reside at the same facility as the netflow source(s) feeding it.
#  UDP packet loss over a WAN will result in substantial loss of flow
#  information.
# This will be especially true during bursts of activity such as a DDOS attack, #
# or network scan.

# If the client is running on a different machine from netflow_server,
# use the ssh command to connect to the netflow server machine and run the
# netflow receiver program, -C (compression) is recommended for WAN use.
# -- CHANGE 127.0.0.1 TO YOUR REAL COLLECTOR MACHINE --
netflow.ssh=ssh -C 127.0.0.1 /opt/sensage/bin/netflow_receiver /opt/sensage/etc/
netflow/netflow_server.prop

# -- OR --

# If the client is running on the same machine as the netflow_server,
# move files directly from the client spool to the server spool when ready.
# Both client and server directories must reside on the same file system.
# This must match the 'netflow.spool' entry in the netflow_server
# configuration file.
netflow.serverSpool=/tmp/netflow_server_spool

# Based on the nfcapd command line, netflow_client will add the -l -t and -p
# options depending on our options
netflow.nfcapd=/opt/sensage/bin/nfcapd
```

Sample netflow_server.prop Configuration File

```
# Logging directory, if specified log files are rotated once
# per day, with 'today.log' containing current log data.
# Otherwise file logging is disabled:
#logging.path=/tmp/netflow_server_logs
logging.path=

# What is the least priority message we log?
#  DEBUG
#  INFO
#  WARNING
#  ERROR
logging.level=INFO

# 1 - enable logging to syslog, 0 - disable:
logging.syslog=1

# our pid/lock file
run.pid=/tmp/netflow_server.pid

# incoming netflow queue directory
netflow.spool=/tmp/netflow_server_spool

# path and any additional arguments for nfdump
# we will add a -r with a directory of files for each call we make
netflow.nfdump=/opt/sensage/bin/nfdump -o "fmt:%ts %te %td %pr %sap -> %dap %sas
%das %in %out %flg %tos %bps %pps %bpp %pkt %byt %fl" -q

## File destination example 'myfile'

# File name for output file if dest.type is 'file'.
# Several replacement options are available:
#  %S if present is replaced by the current epoch second
#  %L if present is replaced by the local YYYYMMDDhhmmss
#  %G if present is replaced by GMT YYYYMMDDhhmmss
# Files while being written have a .wrk extension added to the end
# they are renamed to the path name when done.
dest.myfile.path=/tmp/netflow_collector_logs/netflow%L.log

# seconds between file rotations
dest.myfile.rotate=600

# type file
dest.myfile.type=file

## TCP destination example 'myparser'

# address
dest.myparser.address=127.0.0.1:5014

# type
```

```
dest.myparser.type=tcp

# output queue size in bytes, should be large to prevent data loss during
# a transient network outage or burst.
dest.myparser.queue=10000000
```

CHAPTER 6

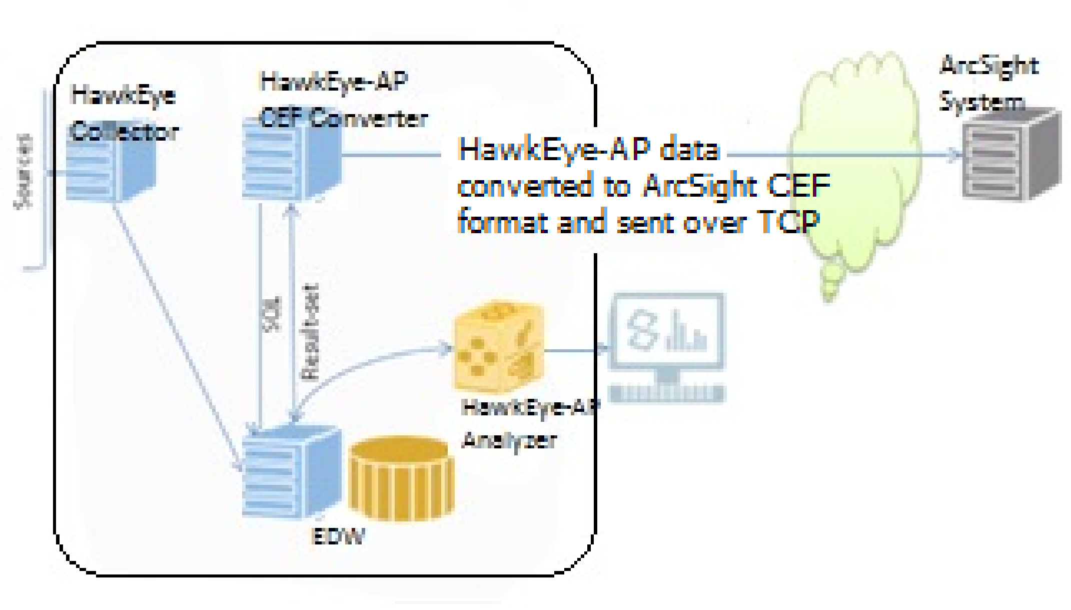
Configuring Common Event Format (CEF) Forwarder

The HawkEye AP CEF Forwarder provides the means to convert HawkEye AP event data to the CEF format and forward this data to ArcSight for real-time processing. The Forwarder is installed with a standard HawkEye AP product installation. Note that the **sensageCef** init script is installed in the same location as other HawkEye AP product init scripts, in the etc/init.d directory.

After configuring CEF, the service must be manually configured to start when the system boots. The sample configuration file is located in [installation-prefix]/etc/cef/example.prop and should be copied to [installation-prefix]/etc/sysconfig/sensageCef.prop.. For set up instructions, refer to "Setting Up HawkEye AP CEF Forwarder", on page 142.

The Forwarder program retrieves data from the EDW and converts it to the CEF format, forwarding the data to the CEF Forwarder via TCP (as shown in Figure 6-1).

Figure 6-1HawkEye AP CEF Forwarder



The CEF Forwarder employs a high-performance C-based query library created for HawkEye Open Access Extension (OAE) to achieve rapid data downloading. Prior to starting an EDW download, the Converter verifies its connection to the CEF Forwarder. The Forwarder logs EDW query SQL strings before a query is run to ensure that in the case a query does not complete (due to system failure), data is retrieved.

If the socket connection to the CEF Forwarder is lost, the Forwarder attempts to complete the data transfer, logging successful batch transfers to syslog as well as any data retrieval or data transfer errors. After logging each query, the Forwarder grabs the Upload IDs, followed by the actual new row data as in the following sample output:

```
2013-07-18T21:40:15.504368Z NOTICE Starting retrieving upload ids from the SLS
2013-07-18T21:40:15.504447Z NOTICE starting SLS query: SELECT uploadid, max_ts,
min_ts, load_count FROM system.upload_info WHERE successful AND load_count > 0
AND
_timediff(max_ts, now()) < 86400000000 AND original_tablename in
```

```
@_list('analytics.bluecoat_sgproxy_batch') DURING ALL;
2013-07-18T21:40:17.179185Z NOTICE finished SLS query, got 21 rows, in 2 seconds
Query for New rows:
2013-07-18T21:40:17.179755Z NOTICE starting SLS query: SELECT ts, uploadid,
cef_version, ...............
```

When stopped by a user or error, the Fowarder logs its state information and exits. State information includes:

Record counts, a list of data upload IDs that fall within the current upload window and show:

- the number of rows present in an uploadID
- the number of rows that are queued for transfer
- the number of rows that were transferred*

* In error conditions the row count may not be completely accurate due to TCP/IP buffering. Because there is no remote application level acknowledgement of records transferred, rows successfully sent to the TCP/IP stack are considered to be successfully transferred.

- **Normal messages** for start and end of transfer, and if applicable, messages that indicate no new Upload IDs are ready--so there is nothing to transfer.
- **Debug messages** include DNS lookups and IP address and note severity.
- **Timestamps** include logging for transfer start, stop and durations and service start, stop and execution durations. For each Upload ID, the Converter provides an ISO microsecond timestamp (UTC) for the last successfully processed data.

For a complete sample listing, see “HawkEye AP CEF Forwarder Sample Log Output”, on page 152.

HawkEye AP CEF Forwarder Multi-Daemon Environment

Multiple sensageCef daemon services are needed when you want to keep track of more than one EDW table separately. In a sensageCef multi-daemon environment, each daemon requires separate configuration files and init script files. The <*configuration_file*> that a customer sets up must have the identical filename as the properties file and the init script file. The state.pid property must also be correctly set in the referenced property file.

For example, a configuration named "Archive" requires the following two files:

```
/opt/sensage/etc/init.d/sensageCef-Archive
/opt/sensage/etc/sysconfig/sensageCef-Archive.prop
```

The sensageCef-Archive.prop property file must have the state.pid value, which is set to:

```
state.pid=/opt/sensage/var/run/sensageCef-Archive
```

For details on adding a new configured daemon to operate with the provided init scripts, see “Adding a sensageCef Daemon to your Configuration”, on page 146.

SPECIFYING HAWKEYE AP CEF FORWARDER OPTIONS

This section contains options for using the HawkEye AP CEF Forwarder. For details on setting up the Converter, see “Setting Up HawkEye AP CEF Forwarder”, on page 142.

Synopsis

The following is the **sensageCef** command line syntax:

```
sensageCef [<options>] {configuration_file}
```

Description

The Converter supports several command options (listed below) for debugging or stopping a process. In normal operations, no options are specified and only the configuration file is executed as in:

```
sensageCef configuration_file
```

Options

You can view an option listing by specifying the **--help | --h** command line option.

Option	Description
-t	Test mode that tests EDW retrieval and outputs CEF messages to stdout; state is not updated.
-v	Records internal operations on stdout.
-c	Runs program as normal in the foreground (no daemonizing) and logs to the console.
-s	Stops current process gracefully.
-S	Forces (kills) the current process to stop.
-H	Converts clear text password to an obfuscated code. **Note:** When a user enters a clear text password, letters will not be diisplayed.

Example

In this example, the **sensageCef** command is executed with the **-H**.

Enter at the command line:

```
> /opt/sensage/bin/sensageCef -H
```

Enter the password to be hidden. The password is not displayed while it is entered. Use the printed, obscured password value in the correct password field of the CEF configuration file.

```
  F68871A4F8C33707234B
```

SETTING UP HAWKEYE AP CEF FORWARDER

Setting up the HawkEye AP CEF Forwarder requires the following steps:

1. Create the "cef" Postgres User with a random password. For details, see "Creating the Postgres User and Schema for CEF Forwarder", next.

2. Create the appropriate table in a postgres instance to track uploads. For details, see "Creating the Postgres Table for Uploads", on page 142.

3. Modify the Collector to track uploads. For details, see "Modifying the Collector", on page 143.

4. Create a configuration file that contains property settings for the Converter. For details, see "Creating Your Configurations File", on page 143.

5. Create a properties file for column mappings. For details, see "Creating Your Column Mappings File", on page 145.

6. When adding a new configured daemon to operate with the provided init scripts, create and execute the init script. For details, see "Adding a sensageCef Daemon to your Configuration", on page 146.

Creating the Postgres User and Schema for CEF Forwarder

Perform the following steps to create the Postgres User (with random password) and schema:

1. Run the following script with the hexis user to create a "cef"postgres user with a random password:

   ```
   /opt/hexis/hawkeye-ap/bin/CEFSetupPostgres.sh
   ```

2. Run the **sensageCef** command (as shown below) to create the hidden password; copy and paste the password displayed in the output of the commands in Step 1 (above).

   ```
   /opt/hexis/hawkeye-ap/bin/sensageCef -H
   ```

3. Run the **sensageCef** command (as shown below) to create a schema in Postgres; use the password displayed in the output of Step 2 (above).

   ```
   /opt/hexis/hawkeye-ap/bin/sensageCef -p "host=<postgres host>
   dbname=controller user=cef password<>"
   ```

Creating the Postgres Table for Uploads

In the following situations, you will need to create the appropriate table in postgres so the Collector can track successful uploads:

- Your default postgres instance does not have the proper table definition.

- For other reasons, you want to track uploads in another postgres instance.

To create the table for the situations noted above, you can use the **sensageCef** executable with the **–p** option. This option takes a connection string and creates the proper table in postgres; for example:

```
./sensageCef –p “host=myhost.com dbname=controller user=controller
password=D4C69546EE84DFD69229E134A69439B1DDB5ABB93698”
```

To obtain a hidden password you run **sensageCef** with the **–H** option; for example:

```
./sensageCef –H
```

After supplying your database password you will get back the obfuscated version which you can then use in your connection string.

Modifying the Collector

NOTE: Please refer to this guide for greater details on overall Collector setup.

After the table in postgres has been created, you need to add a new element to every loader in your **config.xml** file that you will want as a source for the converter. The element, **TrackUploads**, has a single attribute, which is the connection string used to communicate with postgres. So for example if you have a loader that you want to track and use as a source in the converter, you would modify the TrackUploads element to look something like this:

(bold indicates what you need to add to the loader definition)

```
<Loader name="windowsloader" enabled="1">

  <TrackUploads connectionstring="host=myhost.sensage.com dbname=controller
  user=controller password=D4C69546EE84DFD69229E134A69439B1DDB5ABB93698"/>
  <LogQueue>win_logqueue</LogQueue>
  <PTL table="winevt" namespace="addamark.woodshed" type="file">
  <Location>/opt/sensage/etc/collector/agent.ptl</Location>
  </PTL>
  <RunOnHost>localhost</RunOnHost>
  <SLSInstance>newts04.sensage.com:8072</SLSInstance>
  <SLSUser>administrator</SLSUser>
  <SLSPasswd>changeme</SLSPasswd>

</Loader>
```

Now when you restart the collector, all successful loads will get tracked in the postgres database you specified.

Creating Your Configurations File

When you run the converter you supply a configuration file. This file includes several properties that define the behavior of the Converter’s forwarding operation. See Appendix A for a sample configuration file, **example.prop**, provided with the Converter program file.

IMPORTANT: If you are using a multi-service environment, make sure to provide each service its own init script and associated configuration file in the sysconfig directory. For example, if you have three scripts, you are required to provide a separate configuration file for each script in the directory.

- The sensageCef script requires **sensageCef.prop** in the sysconfig directory.
- The sensageCef-optionalConfig script requires **sensageCef-optionalConfig.prop** in the sysconfig directory.
- The sensageCef-extra script requires **sensageCef-extra.prop** in the sysconfig directory

For details, see “Adding a sensageCef Daemon to your Configuration”, on page 146

Attribute Name	Purpose
logging.path	Path where the Converter stores its own log messages.
logging.syslog	Set to **1** to enable logging to syslog; otherwise set to **0** to disable.
state.pid	A fully-qualified pathname to the pid file to store the running process ID of the sensageCEF daemon service.

gen.maxTempFailures	This is the number of consecutive transfer errors to allow before permanently aborting operations. This integer value sets the number of times a transfer failure can occur before CEF exits with a failure status (in which **0** means keep trying). In summary:
	0 = disabled (default)
	1..n - abort after this many failures.
	For example:
	`genmaxTempFailures={0..n}`
	`genmaxTempFailure=2`
	
	Note: Transfer failures are considered to be TCP/IP socket errors. They are a failure to connect, failure to transfer data, and encounter of remote application timeout The transfer will restart after a transfer failure until the error count has been reached. A successful transfer will reset the error count to 0 (zero).

schedule.interval	Specifies how often to run the Converter. You can qualify your number with the following letters: s - seconds m - minutes h - hours d- days For example: Specify **4d** to run every 4 days.
postgres.connectstring	A string (in double quotes) that specifies from where to get upload information; for example: "host=myhost dbname=controller user=controller password=D4C69546EE84DFD69229E134A69439B1DDB5ABB93698"
EDW Settings	
sls.max_age	This is a cutoff. The Converter does not process uploads older than this date. For example, the Converter will only use uploads that have occurred in the last two day when you specify: `sls.max_age=2d`
sls.max_upload_range	This is a cutoff. The converter does not process any upload that spans a range of time greater than this range of time. This feature is used to improve EDW performance. For example, the Converter will only use uploads whose min and max timestamps are no farther than two days apart when you specify: `sls.max_upload_range=2d`
sls.user	The user name used to communicate with the EDW.
sls.pass	The obfuscated password used to connect to the EDW (created with the **sensageCef -H** option).
sls.nodes	A list of EDW hosts (in hostname:port syntax) to communicate with. For example: `sls.nodes=localhost:8072`
sls.table	The fully-qualified name of the table or view used to pull records from.
sls.sourcetables	This is one or more tables that contain the actual data you want to send. For example, assume **sls.table** specifies a view that is a union over **tableA** and **tableB**; then you specify: `sls.sourcetables=tableA,tableB`

Attribute Name	Purpose
sls.max_rows	The maximum number of rows to forward on any given iteration. (This is useful to prevent too much data going to ArcSight at one time.)
sls.max_ids	The maximum number of upload IDs to process at any given time.
sls.where	This is an optional field that can be used to augment the EDW query and narrow the number of records forwarded to ArcSight.
Destination Settings	**You can specify multiple destinations for the data by creating multiple groups of:** **dest.{*NAME*}.{*KEY*}={*VALUE*} pairs**
dest.mynet.term	Row termination character: • **none** - no termination, useful in general only UDP. • **null** - \0 termination, commonly used in syslog over TCP. • **lf** - line feed termination, commonly used readable text output.

dest.mynet.type	Type of destination:
	• **TCP** - send over a TCP socket. Use **dest.**{**NAME**}.*address* to specify a *hostname*:*port* and the *timeout*.
	• **UDP** - send as UDP packets. Use **dest.**{**NAME**}.*address* to specify a *hostname* and the *timeout.*
	• **file** - write to file(s); use dest.{NAME}.path to specify a *pathname.*
	For example:
	`dest.mynet.type=file`
	`dest.mynet.path=/tmp/output%LX%GX%S.log file`

dest.<type>.timeout	This is an integer value in seconds specifying how long the TCP/IP socket connection will remain open without successfully transferring any data. This can be tuned to the expected response times of the destination application. If no data is transferred for at least the amount of time specified, the connection will be terminated and a failure will be logged. To disable the timeout and never detect a non-responsive destination application set this value to 0 (zero). In summary: **0** = disabled (default) **1..n** - enabled. For example: dest.mynet.timeout=30
output.format	Output formatter information: **cef** - Common Event Format **cef.columnsFile** - The column mappings file (See Appendix B for more details.)

Creating Your Column Mappings File

The column mappings file is used to change how columns are mapped from the EDW source table to the Common Event Format. Appendix B provides an example mappings file. You map the columns in your EDW table to the names you want to use in the Common Event Format output. In addition to mapping those columns there are a handful of special column designations that are in the table that follows.

IMPORTANT: When forwarding event data to CEF receiver, the CEF forwarder ignores the columns that have the below mentioned values, which effectively indicate "null" value.

- ""
- "-"
- "-1"

Attribute Name	Purpose
cef.prefix	The output prefix that is attached when rows are formatted.
special.count	specifies how many special pipe-delimited columns should appear at the beginning of the event.
special.{number}	Used to specify speical.{number} values up to the count in speical.count. For example: • special.1=device_vendor • special.2=device_product • special.3=device_version • special.4=signature_id • special.5=name • special.6=severity
special.uploadid	The name of the _uploaded column in the table or view.
special.ts	The name of the ts column in the table or view.

Adding a sensageCef Daemon to your Configuration

To create a new configured daemon that operates with the provided init scripts:

1. Create symlinks for the init scripts in /etc/init.d/ folder as below:

```
lrwxrwxrwx 1 root root 44 Nov 19 05:52 sensageCef-<configuration> -> /opt/
sensage/etc/init.d/sensageCef
```

2. Execute following command to add the new init script to services:

```
chkconfig --add sensageCef-<configuration>
```

3. Create the property file. The property file must be named:

```
/opt/sensage/etc/sysconfig/sensageCef-<configuration>.prop
```

4. Configure the .prop (property) file. The init script uses the installation location.

a Provide a fully-qualified pathname to the lock file. For init script compatibility, set it to:

```
state.pid=/opt/sensage/var/run/sensageCef-<configuration>
```

5. Manage the HawkEye AP Common Event Format Forwarder for the new configuration using the following service commands:

```
service sensageCef-<configuration> start|stop|status|restart
```

SAMPLE CONFIGURATION FILE

You can define parameters for Forwarder operations in a configuration file. The following is a sample configuration file, **example.prop**, provided with the Forwarder program file:

```
# Logging directory, if specified log files are rotated once
# per day, with ’today.log’ containing current log data.
# Otherwise file logging is disabled:
logging.path=/tmp

# 1 - enable logging to syslog, 0 - disable:
logging.syslog=1

# Number of consecutive transfer errors to allow before permanently
# aborting operations.
#  0 = disabled (default)
#  1..n = abort after this many failures
#  gen.maxTempFailures={0..n}
gen.maxTempFailures=0

# How long do we wait between polling?
#  N{s,m,h,d}
#   s - seconds
#   m - minutes
#   h - hours
#   d - days
schedule.interval=10m

# connection string used to get upload information from postgres.
# the password should be the obfuscated version which you can create using the -
H flag to sensageCef
# example:
# postgres.connectstring="host=localhost dbname=controller user=controller
password=D4C69546EE84DFD69229E134A69439B1DDB5ABB93698"

postgres.connectstring="host=localhost dbname=controller user=controller
password=9280D45CDFB53690E13468B5128709DA"

# How many hours old must SLS data be before we ignore it?
#  N{s,m,h,d}
sls.max_age=5d

# sls.max_upload_range is optional, if set it determines the max difference
between
# any uploads ids min_ts and max_ts (to prevent using uploads that cover a huge
time range)
# N{{s,m,h,d}
# for example: sls.max_upload_range=7d  specfieis that any uploadid with a time
range > 7 days will be ignored.  This can
# prevent performance problems for trying to scan large ranges of time

# The sls login information:
sls.user=administrator
# Obscured password generated using -H option
sls.pass=128EF6395F66BBB2E137D5C729FA30A2815E
# SLS Node(s)
sls.nodes=localhost:8072
```

```
# The fully qualified table or view to retrieve from:
sls.table=analytics.microsoft_windows2008_securityEvent_sensageRetriever

# sls.table can refer to a view, so you must also specify the source tables to
use when scanning for new upload ids
sls.sourcetables=analytics.microsoft_windows2008_securityEvent_sensageRetriever

# Maximum rows to retrieve from the SLS per query:
sls.max_rows=100

# Maximum upload ids per query:
sls.max_ids=10

# Optional additional where conditions for the SLS query:
#  For example you might want to restrict records to only those
#  matching a particular criteria.
#  sls.where=bytesIn + bytesOut > 1024*1024
sls.where=

# Multiple destinations of assorted types may be specified of the form:
#  dest.{NAME}.{KEY}={VALUE}
# Where NAME is an arbitrary user assigned destination name

## Network example ’mynet’

# Row termination character:
#  none - no termination, useful in general only with udp
#  null - \0 termination, commonly used in syslog over tcp
#  lf   - line feed termination, commonly used readable text output
dest.mynet.term=lf

# Type of destination:
#  tcp  - send over a tcp socket
#  udp  - send as udp packets
#  file - write to file(s)
dest.mynet.type=tcp

# Destination IP address and port for dest.type ’udp’ or ’tcp’:
# For type ’tcp’ more than one ip:port may be specified. If a
# connection to the first fails, the next will be tried.
dest.mynet.address=localhost:1113
# Timeout for non-responsive sockets with dest.type ’tcp’:
#   value is in seconds:
#   0 = disabled
#   1..n = enabled
dest.mynet.timeout=30

## File example ’myfile’

# File name for output file if dest.type is ’file’.
# Several replacement options are available:
#  %S if present is replaced by the current epoch second
#  %L if present is replaced by the local YYYYMMDDhhmmss
#  %G if present is replaced by GMT YYYYMMDDhhmmss
# Files while being written have a .wrk extension added to the end
# they are renamed to the path name when done.
dest.myfile.path=/tmp/test%LX%GX%S.log

# type file
```

```
dest.myfile.type=file

# line feed terminator
dest.myfile.term=lf

# PID file specification.  If used with the provided init script this
# must be set to:
#    state.pid=/opt/sensage/var/run/sensageCef-<configuration>
# where <configuration> is the trailing part of the init script
# filename.  E.g. for a configuration named "Archive" the following
# two files would be necessary:
#    /opt/sensage/etc/init.d/sensageCef-Archive
#    /opt/sensage/etc/sysconfig/sensageCef-Archive.prop
# and the state.pid value in the prop file must be set to:
#    state.pid=/opt/sensage/var/run/sensageCef-Archive
#
state.pid=/opt/sensage/var/run/sensageCef-win

# Output formatter information:
#  cef - Common Event Format
output.format=cef

# Properties file describing the desired sls->cef column mapping
# application to output.format ’cef’ only. This can be an absolute
# path, or a path relative to this config file.
cef.columnsFile=/opt/sensage/etc/cef/wincefColumns.prop
```

HawkEye EDW to ArcSight CEF Column Name Mappings

The following is a listing of all HawkEye EDW column names, mapped to the corresponding ArcSight CEF column names. Columns can be added or changed without recompiling the binary code. This listing is contained in the **cefColumns.prop** file provided with the Converter software.

```
# The output prefix we attach when formatting rows:
cef.prefix=SenSageCEF_Forwarder:

# The special initial pipe delimited columns and the SLS column name #
associated with them:
special.count=7
special.0=cef_version
special.1=device_vendor
special.2=device_product
special.3=device_version
special.4=signature_id
special.5=name
special.6=severity

# The name of the _uploaded column and ts in our table or view:
special.uploadid=uploadid
special.ts=ts


# List of SLS column names with an optional alternative name for # CEF
name/value pair output:
column.ApplicationProtocol=app
column.baseEventCount=cnt
column.bytesIn=in
column.bytesOut=out
column.destinationAddress=dst
column.destinationHostName=dhost
column.destinationMacAddress=dmac
column.destinationNtDomain=dntdom
column.destinationPort=dpt
column.destinationProcessName=dproc
column.destinationUserId=duid
column.destinationUserName=duser
column.destinationUserPrivileges=dpriv
column.deviceAction=act
column.deviceAddress=dvc
column.deviceCustomNumber1=cn1
column.deviceCustomNumber1Label=cn1Label
column.deviceCustomNumber2=cn2
column.deviceCustomNumber2Label=cn2Label
column.deviceCustomNumber3=cn3
column.deviceCustomNumber3Label=cn3Label
column.deviceCustomNumber4=cn4
column.deviceCustomNumber4Label=cn4Label
column.deviceCustomNumber5=cn5
column.deviceCustomNumber5Label=cn5Label
column.deviceCustomNumber6=cn6
column.deviceCustomNumber6Label=cn6Label
column.deviceCustomString1=cs1
column.deviceCustomString1Label=cs1Label
column.deviceCustomString2=cs2
column.deviceCustomString2Label=cs2Label
column.deviceCustomString3=cs3
column.deviceCustomString3Label=cs3Label
column.deviceCustomString4=cs4
column.deviceCustomString4Label=cs4Label
column.deviceCustomString5=cs5
column.deviceCustomString5Label=cs5Label
column.deviceCustomString6=cs6
column.deviceCustomString6Label=cs6Label
column.deviceEventCategory=cat
column.deviceHostName=dvchost
column.endTime=end
column.fileName=fname
column.fileSize=fsize
column.message=msg
column.receiptTime=rt
column.requestURL=request
column.SourceAddress=src
column.sourceHostName=shost
column.sourceMacAddress=smac
column.sourceNtDomain=sntdom
column.sourcePort=spt
column.sourceUserId=suid
column.sourceUserName=suser
column.sourceUserPrivileges=spriv
column.startTime=start
column.transportProtocol=proto
```

HawkEye AP CEF Forwarder Sample Log Output

```
2013-06-21T00:08:41.844121Z NOTICE Starting retrieving upload ids from the
SLS
2013-06-21T00:08:41.844170Z NOTICE starting SLS query: SELECT uploadid,
max_ts, min_ts, load_count FROM system.upload_info WHERE successful AND
load_count > 0 AND _timediff(max_ts, now()) < 864000000000000 AND
original_tablename in @_list('analytics.bluecoat_sgproxy_batch')  DURING
ALL;
2013-06-21T00:08:43.083386Z NOTICE finished SLS query, got 9 rows, in 2
seconds
2013-06-21T00:08:43.083425Z INFO Reading state data
2013-06-21T00:08:43.083499Z NOTICE Finished retrieving upload ids from the
SLS
2013-06-21T00:08:43.083951Z NOTICE starting SLS query: SELECT ts, uploadid,
cef_version, device_vendor, device_product, device_version, signature_id,
name, severity, ApplicationProtocol, SourceAddress, baseEventCount, bytesIn,
bytesOut, destinationAddress,  destinationDnsDomain, destinationHostName,
destinationMacAddress, destinationNtDomain, destinationPort,
destinationProcessName, destinationServiceName,
destinationTranslatedAddress, destinationTranslatedPort, destinationUserId,
destinationUserName, destinationUserPrivileges,  deviceAction,
deviceAddress, deviceCustomDate1, deviceCustomDate1Label, deviceCustomDate2,
deviceCustomDate2Label, deviceCustomNumber1, deviceCustomNumber1Label,
deviceCustomNumber2, deviceCustomNumber2Label, deviceCustomNumber3,
deviceCustomNumber3Label,  deviceCustomNumber4, deviceCustomNumber4Label,
deviceCustomNumber5, deviceCustomNumber5Label, deviceCustomNumber6,
deviceCustomNumber6Label, deviceCustomString1, deviceCustomString1Label,
deviceCustomString2, deviceCustomString2Label, deviceCustomString3,
deviceCustomString3Label, deviceCustomString4, deviceCustomString4Label,
deviceCustomString5, deviceCustomString5Label, deviceCustomString6,
deviceCustomString6Label, deviceDirection, deviceDnsDomain,
deviceEventCategory, deviceExternalId, deviceFacility,  deviceHostName,
deviceInboundInterface, deviceMacAddress, deviceNtDomain,
deviceOutboundInterface, deviceProcessName, deviceTranslatedAddress,
endTime, externalId, fileCreateTime, fileHash, fileId, fileModificationTime,
fileName, filePath, filePermission,  fileSize, fileType, message,
oldFilename, oldfileCreateTime, oldfileHash, oldfileId,
oldfileModificationTime, oldfilePath, oldfilePermission, oldfileType,
oldfsize, receiptTime, requestClientApplication, requestCookies,
requestMethod, requestURL, sourceDnsDomain,  sourceHostName,
sourceMacAddress, sourceNtDomain, sourcePort, sourceServiceName,
sourceTranslatedAddress, sourceTranslatedPort, sourceUserId, sourceUserName,
sourceUserPrivileges, startTime, transportProtocol FROM
analytics.intellischema.arcsight_cef WHERE  uploadid IN
@_list('599311E4C14ADABDC6E87AFF535EAFB1', 'X') DURING
time('2011-05-13T11:02:12.000000Z'), time('2011-05-13T11:06:13.000000Z');
2013-06-21T00:08:43.107804Z DEBUG parseAddress("e31.sensage.com:1111
<http://e31.sensage.com:1111> ")
2013-06-21T00:08:45.251654Z NOTICE finished SLS query, got 101 rows, in 2
seconds
2013-06-21T00:08:45.252035Z INFO Saving state data
```

APPENDIX A

PTL File Format

Synopsis

```
# optional comments, e.g. example lines
^<regular_expression>|(.*)$
<field_name>:<data_type>[,<field_name>:<data_type>[...]]
-- SQL statement follows from here
SELECT <expression> as ts,          -- first target must be a timestamp named ’ts’
       <expression> [as <column>]  -- any number of target columns may follow
  FROM stdin
;
```

Description

PTL files define how to parse source log entries into fields of information, transform the fields if necessary, and load them into tables in the EDW. The acronym PTL stands for “parse, transform, and load.” Loading operations break log files into single-line log entries, parse the lines into fields, and load the parsed fields into target columns of a table.

Fields to be captured from source log entries are delimited by parentheses in the regular expression on the first line of the PTL file. The fields are named and their data types are declared on the second line. The remaining lines of the PTL file contain a HawkEye AP SQL SELECT statement, which defines the target schema of tables loaded using the PTL file.

PTL files recognize two styles of comment lines. The comment style to use depends on the section of the PTL file.

- The first style uses a number sign (#) at the beginning of comment lines. Use the first style for all comments lines before Line 1, the line with the regular expression.
- The second style uses two dashes (`--`) at the beginning of comments. Use the second style for all comment lines in the SQL section after Line 2, the line with field declarations.

NOTE: Comment lines are not allowed between Line 1 and Line 2 or after the terminating semi-colon of the SELECT statement.

PREREQUISITES

To most effectively use this tutorial, you should have the prerequisite skills, have access to a regular expression editor, and have access to a running HawkEye AP system.

The skills you need to develop PTL files include:

- Writing and debugging regular expressions
- Programming with Perl
- Use of some command-line tools for working with the HawkEye EDW
- Use of HawkEye AP SQL

In addition, you should be familiar with the log sources for which you are developing PTL files.

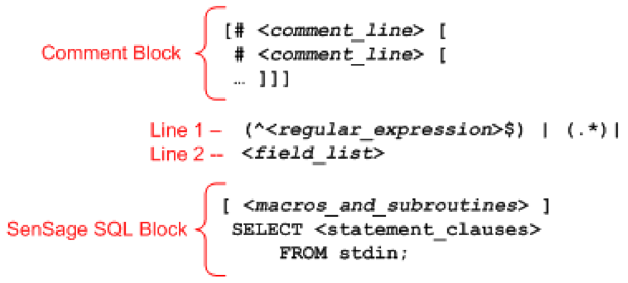
OVERVIEW OF A PTL FILE

This section covers the following topics:

- "Synopsis of a PTL File", next
- "The Sections of a PTL File", on page 155
- "Regular Expression", on page 156

Synopsis of a PTL File

The following syntax diagram illustrates the anatomy of a PTL file:



```
[# <comment_line> [
 # <comment_line> [
 … ]]]

(^<regular_expression>$) | (.*)|
<field_list>

[ <macros_and_subroutines> ]
 SELECT <statement_clauses>
     FROM stdin;
```

The syntax for `<field_list>` in Line 2 is the following:

`<field_name>:<datatype>[,<field_name>:<datatype>[...]]`

The values allowed for `<datatype>` are:

Data Type	Description
BOOL	A logical value indicating true (1) or false (0) stored as 1 byte
FLOAT	A 64-bit floating point value [-1.797693e+308 to 1.797693e+308]
INT32	A signed 32-bit integer value [-2147483647 to 2147483648]
INT64	A signed 64-bit integer value [-9223372036854775808 to 9223372036854775807]

Data Type	Description
TIMESTAMP	A scaled 64-bit floating point value for the integral number of microseconds since 1/1/1970 GMT
VARCHAR	A string with the maximum length of 2147483646 bytes, which is two bytes less than two gigabytes.
INET	A 2, 6, or 18-byte value that represents an IP address for a host or network.The textual representaton of inet includes the subnet sizes, for example 10.56.127.4/24 or 2002::a509:4482/64. If the inet is a host only, then the subnet (netmask) is 32 (IPv4) or 128 (IPv6), representing a single host.

The Sections of a PTL File

A PTL file comprises three functional sections, which occur in this order:

1 A **regular expression**, which describes how to parse fields from source log entries

2 A set of **field names** and **data types** for the fields that are parsed by the regular expression

3 A **SELECT statement** with target columns that are the columns of tables to be loaded

Consider the following example:

```
*([^ ]+) +([^ ]+) +([^ ]+) +\[([0-9]+/[a-z]+/[0-9]+):([0-9]+:[0-9]+:[0-9]+) +([^
]+) +([^ ]+) +([^ ]+) +([^ ]+) +([^ ]+) +([^ ]+)$
ClientIP:VARCHAR,UserAgent:VARCHAR,UserName:VARCHAR,Date:VARCHAR,Time:VARCHAR,
Method:VARCHAR,Url:VARCHAR,HttpVers:VAR,RespCode:INT32,RespSize:INT32
SELECT _timeparse( _strcat( Date, " ", Time), "%Y-%m-%d %H:%M:%S") as ts,
       ClientIP, UserAgent, UserName, Method, Url, HttpVers, RespCode, RespSize
  FROM stdin
;
```

The First Line

The first line contains an entire regular expression, with parentheses that define *capture groups*:

```
*([^ ]+) +([^ ]+) +([^ ]+) +\[([0-9]+/[a-z]+/[0-9]+):([0-9]+:[0-9]+:[0-9]+)
+([^ ]+) +([^ ]+) +([^ ]+) +([^ ]+) +([^ ]+) +([^ ]+)$
```

A capture group is a portion of a regular expression enclosed within parentheses. It isolates the portion of the regular expression that describes the matching pattern for a specific field of information within source log entries. When a portion of a log entry matches the pattern in a capture group, that portion of the log entry is captured in memory for later use. Captured portions are stored in fields that you declare on the second line of the PTL file.

The Second Line

The second line contains field declarations, which give names to capture groups and specify their data types:

```
ClientIP:VARCHAR,UserAgent:VARCHAR,UserName:VARCHAR,Date:VARCHAR,Time:VARCHAR,
Method:VARCHAR,Url:VARCHAR,HttpVers:VAR,RespCode:INT32,RespSize:INT32
```

The field declarations on the second line correspond in number and order to the capture groups in the regular expression.

The Remaining Lines

The remaining lines of a PTL file contain a HawkEye AP SELECT statement:

```
SELECT _timeparse( _strcat( Date, " ", Time), "%Y-%m-%d %H:%M:%S") as ts,
       ClientIP, UserAgent, UserName, Method, Url, HttpVers, RespCode, RespSize
  FROM stdin
;
```

PTL files end with the terminating semi-colon (;) of the SELECT statement. If you include any text, such as comment lines, following the semi-colon, your PTL file will fail to load log entries successfully.

Regular Expression

The regular expression on the first line of a PTL file parses similarly structured log entries into the fields of information declared on the second line.

As an example, let's look at the head of the `syslog_log` source data file. HawkEye AP provides this sample file in `example_logs/syslog`. To list the sample log file, run the following command:

```
gunzip -c syslog_log.gz | head -20
```

The output is:

```
Sep 23 04:02:00 a05-a syslogd 1.4-0: restart.
Sep 23 04:02:00 a05-a syslogd 1.4-0: restart.
Sep 23 04:02:00 a05-a syslogd 1.4-0: restart.
Sep 23 04:02:00 a05-a syslogd 1.4-0: restart.
Sep 23 04:02:00 a05-a syslogd 1.4-0: restart.
Sep 23 05:00:12 a05-a sshd[31380]: Connection closed by 10.0.1.111
Sep 23 06:00:12 a05-a sshd[31398]: Connection closed by 10.0.1.111
Sep 23 07:00:11 a05-a sshd[31416]: Connection closed by 10.0.1.111
Sep 23 08:00:11 a05-a sshd[31434]: Connection closed by 10.0.1.111
Sep 23 09:00:11 a05-a sshd[31452]: Connection closed by 10.0.1.111
```

Analyzing Source Log Entries

Before you write the regular expression, carefully analyze the source log entries to determine what information they contain and how the information is formatted. A single log entry from the example source log file looks like the following:

```
Sep 23 04:02:00 a05-a syslogd 1.4-0: restart.
```

The entry contains six items of information:

- Month—`Sep`
- Day—`23`
- Time—`04:02`
- Host Name—`a05-a`
- Program—`syslogd 1.4-0`
- Message—`restart.`

Notice that the year in which the log entry was written is not recorded. Log entries require a timestamp that includes the year in order to be loaded successfully. In this case, we have to implement a PTL file that lets someone specify the year for timestamps during loading operations.

Notice also that some entries have spaces within the program field, while others do not:

```
syslogd 1.4-0
sshd[31380]
```

Our regular expression must address the issue of embedded spaces within a field.

Identifying Capture Groups and Fields

After you analyze source log entries, you can identify the capture groups you want in the regular expression and their corresponding field declarations on Line 2. To parse log entries from our sample source log file, the regular expression will specify five capture groups for these fields:

(timestamp) (host) (program) (process ID) (message)

Capturing the Timestamp

For the (timestamp) field, we create a capture group in the regular expression that matches the first three characters, then a space, then the next two digits, and then the next eight characters. For flexibility, we write the capture group to match any number of month characters. It accepts `S`, `Se`, `Sep`, `Sept`, or `September`. Some source log entries might separate the month and day with more than one space, so the regular expression accepts multiple, consecutive spaces. Also, it accepts the day as `011`, or `11`, or `eleven`.

A more flexible way to write the capture group for (timestamp) is to say: “Add each consecutive character to the captured field until you come across a space; then after one or more consecutive spaces, add one space to the captured field; then add each consecutive character until you find a sequence of spaces and add one space to the captured field; then add each consecutive character until you find a space; then end the field.”

In a regular expression, a capture group that corresponds to the statement above looks like this:

```
(\S+\s+\S+\s+\S+)
```

where:

Regular Expression Patterns	Meaning
`(`	Start a capture group
`\S`	Match any character other than a space or a tab
`+`	Repeat the prior pattern, `\S`, as long as it matches at least once
`\s`	Match the space character
`+`	Repeat the prior pattern, `\s`, as long as it matches at least once
`\S`	Match any character other than a space
`+`	Repeat the prior pattern, `\S`, as long as it matches at least once
`\s`	Match the space character
`+`	Repeat the prior pattern, `\s`, as long as it matches at least once
`\S`	Match any character other than the space character
`+`	Repeat the prior pattern, `\S`, as long as it matches at least once

Regular Expression Patterns	Meaning
)	End the capture group

Next we need to match the spaces that delimit the (timestamp) field from the (host) field, but we do not put them within the parentheses of a capture group:

```
\s+
```

Delimiting spaces in the source log entry are skipped over and not included in the fields used in the SELECT statement.

Capturing the Host

For the (host) field we need to match all characters other than spaces. In a regular expression, such a capture group look like this:

```
(\S+)
```

We follow the capture group for (host) with a match for one or more delimiting spaces to skip:

```
\s+
```

Capturing the Program

Capturing the (program) field and the (process id) field is tricky. We cannot use spaces as the delimiter at the end of the (program) field, because program names can include spaces. We cannot use another, single character as the delimiter, either. The (program) field ends with a left square bracket (`[`) when there is a process ID, like this: `'sshd[31434]:'`, but the fields ends with a colon (`:`) when the (process id) field is missing, like this: `'syslogd 1.4-0:'`.

To handle these issues, we first create the (program) capture group with a *character class* that defines the characters to match. A character class is defined between square brackets. To capture the (program) field, our character class says, “Match any characters except a left square bracket ([) or a colon (:).” A left square bracket delimits the end of the (program) field when a process ID is present; a colon (:) delimits the end when the process ID is missing.

Our capture group looks like this:

```
([^[:]+)
```

where:

Regular Expression Patterns	Meaning
(	Start a capture group
[	Start a character class
^	Match any characters except those in this character class
[	Matching ends if a left square bracket in encountered **NOTE:** Generally, you must escape the special meaning of left square brackets in order to match them, but you do not need to escape them within a character class.

Regular Expression Patterns	Meaning
`:`	Matching ends if a colon is encountered
`]`	End the character class
`+`	Repeat the prior pattern, `[^[:]`, as long as it matches at least once
`)`	End the capture group

After reaching the end of the (program) field, we look for a left square bracket (`[`), which indicates the presence of an optional process ID. Outside a character class, left square brackets are metacharacters, so we must escape their special meaning with a preceding backslash, like this: `'\['`. Because the process ID and its left bracket might not be present, we follow the bracket with a question mark (`?`), which indicates that the delimiter is optional. Because it is a delimiter that we want to skip over, we do not enclose it in the parentheses of a capture group.

The optional delimiter between (program) and (process ID) looks like this:

```
\[?
```

Capturing the Optional Process ID

Next we need a capture group for the (process ID) field, if present. Because process IDs are numbers, we can use the special character class for digits, \d. This character class matches any of the arabic numerals 0 through 9.

We follow the digits class with an asterisk (*) to repeat the pattern if there is at least one character that matches. Unlike the plus sign (+), which also indicates to repeat a pattern, an asterisk is an optional match; with a plus sign, the pattern must match at least one character. We use an asterisk in this case, because there are no digits when the process ID is missing.

Because the entire capture group for the (process ID) field is optional, we use a question mark immediately after the capture group to indicate that the numeric sequence is optional. The complete capture group for (process ID) looks like this:

```
(\d*)?
```

Next we match the optional right square bracket that delimits the end of the (process ID) field, if one is present. We must escape the regular meaning of right brackets with a preceding backslash (`\`) in order to match them. We follow the right bracket with a question mark to indicate that the match is optional:

```
\]?
```

Next we match the colon (`:`) that delimits the end of the (program) and optional (process ID) fields. Although the sample log entries indicate that a colon is always present, we use a question mark to indicate it is optional, in case some log entries to do not include it:

```
:?
```

Capturing the Message

Next we look for spaces that delimit the (message) field from the earlier parts of the log entry. We use an asterisk (`*`) in cases where there are no spaces preceding the message or the log entry has no message:

```
\s*
```

Finally, we match any characters for the (message) field. Because it may not be present, we use a dot (`.`) to match any non-null character. We follow it with an asterisk to indicate zero or more occurrences:

```
(.*)
```

IMPORTANT: Only use the "dot star" (.*) pattern at the end of a regular expression. When "dot star" occurs earlier in an expression, performance issues occur due to excessive backtracking by the regular expression engine.

Completing the Entire Regular Expression

Putting this all together, we have the following regular expression, with five capture groups:

```
(\S+\s+\S+\s+\S+)\s+(\S+)\s+([^[:]+)\[?(\d*)?\]?:?\s*(.*)
```

Because we want to start matching at the beginning of each log entry, we enclose the regular expression in parenthesis, add a caret (`^`) at the start, and a dollar sign ($) at the end. These indicate that the regular expression runs from the start of the log entry all the way to the end:

```
^((\S+\s+\S+\s+\S+)\s+(\S+)\s+([^[:]+)\[?(\d*)?\]?:?\s*(.*))$
```

Because we want to load all log entries, even when the match fails, we add an alternative expression that matches any non-null characters:

```
^((\S+\s+\S+\s+\S+)\s+(\S+)\s+([^[:]+)\[?(\d*)?\]?:?\s*(.*))$|(.*)
```

Ellipses as a Parsing Shorthand

The HawkEye AP Perl interpreter, `atperl`, has a proprietary extension that gives special meaning to three successive periods, or ellipses. Generally, a period in a regular expression matches any character. You can place several periods in sequence to indicate how many characters of any kind you want to match.

With HawkEye AP Perl, three periods in a sequence means, "match any number of any characters up to the character or character class that follows." Also, three periods in a sequence implicitly define a capture group; enclosing parentheses are not required. The following example matches any characters until a comma (,) is encountered:

```
...,
```

The ellipses extension is useful for very simple log entries that have a fixed number of fields that are separated by delimiting characters. For example, the following log entry is generated by a hypothetical log source:

2006-06-28T12:15:32+8 elena host-a1 login

All log entries from this log source have the same format, which includes the fields (timestamp), (user), (host), and (action). The fields are separated by single spaces. With the ellipses extension, you could write the following regular expression:

```
^(... ... ... ...)$|(.*)
```

Without the extension, you would write the following, equivalent, standard regular expression:

```
^(([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*))$|(.*)
```

NOTE: In the preceding example of a standard regular expression, we use a custom character class, `[^ ]` in place of the special character class for non-white space characters, `\S`. The character class `\S` excludes white-space characters in addition to spaces, such as tabs. The `[^ ]` custom character class excludes only spaces.

Additional Resources

For full details on creating regular expressions, see:

- *Mastering Regular Expressions* by Jeffrey Friedl, published by O'Reilly and Associates.

  http://www.oreilly.com/catalog/regex/

- The *regex(7)* man page

Creating the PTL

This section includes the following topics:

- "Declaring the Regular Expression", next
- "Field Declarations", on page 162
- "SELECT Statement", on page 163
- "Table Names", on page 166
- "The FROM Clause", on page 166
- "Where Clauses", on page 166

Declaring the Regular Expression

The first functional line in a PTL file, Line 1, is a regular expression. A basic regular expression in a PTL file has the following syntax:

```
^(<regular_expression>)$
```

The caret symbol (ˆ) anchors the evaluation at the beginning of the log entry. The dollar sign ($) anchors the evaluation at the tail end of entry. The parentheses instruct the evaluation to put values that match `<expression>` in the corresponding field on Line 2.

The basic syntax for the regular expression has a problem. It allows parse failures. A fool-proof regular expression that matches any line is:

```
.*
```

You should include this "catch-all" regular expression in PTL files you want to certify. A refined basic syntax for the regular expression has this form:

```
(^<regular_expression>$)|(.*)
```

The OR operator (|) combines two regular expressions. Evaluation begins with the first expression. If the log entry matches the expression, the entire, unaltered source log entry is placed in the first field on Line 2. Otherwise, evaluation proceeds with the second expression. The second expression, `(.*)`, matches any line. If the main expression fails, the catch-all expression captures the log entry and puts it in the second field on Line 2.

Field Declarations

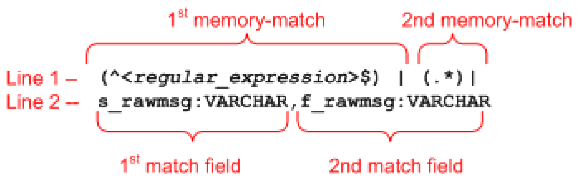
The field declarations on the second line of a PTL file assign names and data types to the capture groups in the regular expression. Data types can be bool, float, int32, int64, inet, timestamp, varchar, or lvarchar. Use the names you declare on Line 2 in the SELECT statement that follows it.

For more information on SQL data types, see Data Types in Chapter 3: HawkEye AP SQL in the *Event Data Warehouse Guide*.

NOTE: Some data type assignments can cause a loss of information; for example, assigning an int64 to an int32 can cause a loss of precision. Other examples include assigning an int32 or int64 data type to a number with a decimal value, assigning a numeric data type to a varchar, etc.

The second functional line in a PTL file, Line 2, is a field list. Fields in the list correspond to memorized matches defined in the regular expression. Fields on Line 2 are similar to the `$1`, `$2`, etc, variables used in Perl script to access memorized matches. The fields in Line 2 capture memorized matching values found between successive pairs of parentheses.

For example, the basic syntax for the regular expression on Line 1 defines two memorized matches. The field list on Line 2 should have two corresponding match fields, as the following example shows:



```
(^<regular_expression>$) | (.*)|
s_rawmsg:VARCHAR,f_rawmsg:VARCHAR
```

You define additional memory matches as you refine your version of *`<regular_expression>`*. Your objective is to parse information out of source log entries. For each memory match you add to your regular expression, you add a corresponding field to the list.

A match field has the following syntax:

<field_name>:{VARCHAR|TIMESTAMP|INET|INT64|INT32|BOOL|FLOAT}

Generally, define match fields as `VARCHAR` types on Line 2, and typecast them later when you load their values into the table.

For more information on the different field types, see "Synopsis of a PTL File", on page 154.

The specification for the field list on Line 2 has the following syntax:

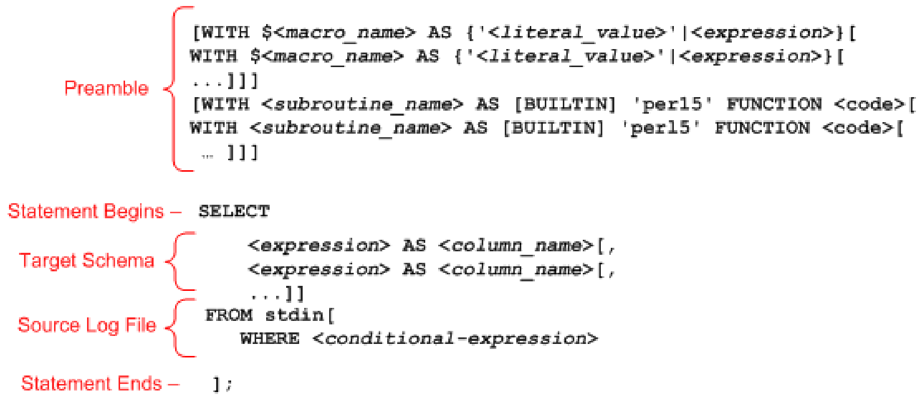
```
s_rawmsg:VARCHAR[,<match_field>[,<match_field>[...]]],f_rawmsg:VARCHAR
```

All certified PTL files have the match fields `s_rawmsg` and `f_rawmsg`. These fields correspond to recommended minimum for Line 1. The first regular expression puts matching log entries in `s_rawmsg`. Failing a match, the second regular expression, the "catch-all," puts unrecognized log entries in `f_rawmsg`. Regardless the success or failure of the PTL file to recognize a log entry, the source log entry is available for loading into the table.

SELECT Statement

The SELECT statement transforms the parsed fields into a list of target columns. The target columns describe the schema for tables loaded with the PTL file.

The SELECT statement in a PTL file has the following syntax:



```
[WITH $<macro_name> AS {'<literal_value>'|<expression>}[
WITH $<macro_name> AS {'<literal_value>'|<expression>}[
...]]]
[WITH <subroutine_name> AS [BUILTIN] 'perl5' FUNCTION <code>[
WITH <subroutine_name> AS [BUILTIN] 'perl5' FUNCTION <code>[
 … ]]]

SELECT
      <expression> AS <column_name>[,
      <expression> AS <column_name>[,
      ...]]
 FROM stdin[
    WHERE <conditional-expression>
  ];
```

For more details on SELECT statements, see "Overview of AP SQL SELECT Statements", on page 41 in the *Event Data Warehouse Guide*.

In a PTL SELECT statement, the column list is called the *target schema*. The columns in the list are the columns in the table to be loaded. Target schemas define the names of columns, their types, and the values to load.

The target schema in the PTL must match the schema of the target table, or the load operation will fail. Column names and column data types must be the same. If you omit table columns from the target schema, or the target schema includes columns not in the target table, the schemas do not match and the load operation fails.

A column specification in the target schema has the following syntax:

```
<expression> AS <column_name>
```

An `<expression>` is anything that evaluates to a scalar value.

A `<column_name>` is the name of a column in the target table. Compose the name with alphanumeric characters and underscores (_).

Specifying Column Names

In the standard SELECT statement, column names are case-insensitive. The same is true of SELECT statements in PTL files, provided the file is used to load an existing table. For example, if the column name is USER_ID in the target table and you specify user_id in the target schema, the columns are considered to match.

Case-sensitivity becomes an issue when PTL files create tables that do not exist. If a PTL file specifies `USER_ID` as a column name as in the target schema, the created table has a column named `USER_ID`. If the PTL file specifies `user_id`, the created table has a column named `user_id`.

If you query a table with `*` in the FROM clause, column headings in the query results have the same case as column names in the table. The case of physical column is also visible when you query system tables for lists of columns.

A general recommendation is to define the names of columns that contain values parsed from log entries in lower case. Define the names of columns that contain values not found directly in log entries in upper case.

Specifying Column Types

In a standard SELECT statement, there is no way to declare the data types of columns in the column list. The data types are defined in the target tables themselves, so there is no need to. Likewise, the SELECT statement in a PTL does not have a way to declare data types for columns in its target schema.

This is not a problem when a PTL file loads an existing table. The column types are defined in the target table. If the expression in a column specification evaluates to different scalar type than the column in the target table, automatic type conversion occurs. For example, the target schema has the following column definition

```
'5' AS login_failure_count
```

The existing table defines `login_failure_count` as type INT64. The expression `'5'` evaluates to a VARCHAR type. The system converts '5' to an INT64 and loads the number into the column.

The problem occurs when a PTL file creates the table it is loading. The data types of columns in created tables are inferred from the scalar types of the evaluated expressions. In the example above, the expression has VARCHAR as its type. Therefore, `login_failure_count` in the created table will also have VARCHAR as its data type.

You can enforce column types in created tables if you typecast the complete expression. The typecast becomes the type of a column when tables are created. The following example shows how to ensure that `login_failure_count` is created as INT64.

```
_INT64('5') AS login_failure_count
```

This example is trivial. Generally, you would not use a literal value in a column specification. A literal was used purely to illustrate the issue of type declaration in PTL files.

Always typecast the expressions in column specifications to the type you want in tables created while loading. In the example above, typecast the expression to ensure that the column is created with INT64 as its type.

To avoid certification failure, PTL files have these column specifications in their target schemas:

```
SELECT
<expression> AS ts,
<expression> AS AUDIT_PARSE_SUCCESS,
<expression> AS unparsed_message,
-- These columns reserved for internal use
_BOOL(0) AS _internal_bool,
_FLOAT(0) AS _internal_float,
_INT32(0) AS _internal_int32,
_INT64(0) AS _internal_int64,
_TIMESTAMP(0) AS _internal_timestamp,
_VARCHAR(“) AS _internal_varchar,
_INET (0) AS _internal_inet,
From stdin;
```

The topic “Storing Raw Log Entries” later in this chapter explains how to handle the expressions for the `AUDIT_PARSE_SUCCESS` and `unparsed_message` columns.

Including the TS Column in Target Schemas

Tables in the EDW are optimized for storing and querying log entries. Part of the optimization requires that log entry tables have timestamp columns named `ts`, with TIMESTAMP as the type. Source log entries have timestamps embedded somewhere, otherwise they are not log entries.

The minimal target schema in a PTL file has the following syntax:

```
WITH <expression> AS ts
```

Without a column specification for the `ts` column, PTL files cannot successfully load or create tables. Generally, *`<expression>`* resolves to the timestamp parsed from the source log entry. The next chapter, “Parsing,” explains how to handle timestamps found in log entries and how to create expressions that successfully load them into `ts`.

PTL files fail certification if they do not store raw log entries in a column named `unparsed_message`. They fail certification if they do not include a column named `AUDIT_PARSE_SUCCESS` that indicates whether the raw log entry was successfully recognized and parsed. Finally, HawkEye AP requires that PTL files declare some special columns reserved for internal use.

Functions in PTL Select Statements

As with standard HawkEye AP SELECT statements, PTL files can include built-in functions and user-defined subroutines.

To learn about built-in HawkEye AP SQL functions, see “Overview of AP SQL SELECT Statements”, on page 41 in the *Event Data Warehouse Guide*.To learn about user-defined functions, see “User-Defined Subroutines”, on page 79 and “Perl Subroutines”, on page 217 in the *Event Data Warehouse Guide*.

Unsupported Keywords in PTL Select Statements

In PTL files, SELECT statements cannot include UNION, GROUP BY, ORDER BY, or DURING clauses, nor can they include subqueries. The EXPLODE keyword is supported; it creates multiple EDW rows for each source log entry.

Timestamps in PTL Select Statements

In PTL files, the first target column must be a timestamp named `ts`. The EDW requires it. Timestamps are stored internally as a number of microseconds since January 1, 1970, GMT. Generally, source log entries contain a field that records when the entry was written.

The recommendation for parsing and loading timestamps is to declare the capture field on line 2 as a varchar data type. Use `_timeparse()` to create the timestamp from formatted strings or `_timestamp()` to convert numeric values to timestamps. Often you apply the `_timeparse()` function in the target column, as the next example shows:

```
...
ts:VARCHAR,...
...
SELECT _timeparse(ts, "%Y-%m-%d %H:%M:%S") as ts,
...
```

For more information on conversion to timestamps, see "Conversion Expressions", on page 72 in the *Event Data Warehouse Guide*.

Schema Changes in PTL Select Statements

IMPORTANT: In preparing for schema changes in the PTL SELECT statement, the practice of creating dummy columns, one for each data type, in PTL scripts is now deprecated. The dummy columns were only required to facilitate the operations of the `CLCHGSCHEMASCRIPT`, which is also now deprecated beginning with Release 6.0.

In Release 6.0, the ALTER TABLE statement subsumes the operations that used to be performed by the `CLCHGSCHEMA` script . For details on the ALTER TABLE statement see "Modifying Table Schema - altertbl", on page 119 in the *Administration Guide.*

Table Names

Table names are handled externally from the statement. The `atload` command manages the loading operation, and it takes the name of the target table from its command line. The following example shows how:

```
atload "host01:1680,host02:1680,host03:1680" my_space.my_table my_ptl_file.ptl \
  my_log.log
```

The FROM Clause

In standard SQL, the FROM clause identifies the source tables. In a PTL file, the FROM clause identifies the source is a single line of text from a log file. The FROM clause has this syntax:

```
FROM stdin;
```

Where Clauses

PTL files can include WHERE clauses to filter out log entries that match the regular expression but which you do not want to load. In general, you should load all log entries. Use WHERE clauses only to skip over empty lines in source log files.

To skip blank log lines, use a conditional expression on one of the parsed fields that you know always contains a value and compare it to an empty string. For example, you might use the timestamp field from line 2, as the next example WHERE clause shows.

```
WHERE timestamp = ''
```

Skipping over blank lines in source log files avoids parse failures and bad loads.

Using Macros

The preamble of the SELECT statement lets you declare macros. A macro defines a value or expression for use later in the preamble and in the SELECT statement. Before compiling the SELECT statement, a preprocessor expands macros, replacing them with their full specification.

NOTE: The preamble also lets you declare custom Perl subroutines.

Synopsis of Macros

A macro declaration in a PTL file has the following syntax:

```
WITH $<macro_name> AS {'<literal_value>'|<expression>}
```

A <macro_name> can have alphanumeric characters and underscores (_). You can declare scalar macros in a PTL file but not list or star macros. Include the dollar sign ($) in references to macros.

A '*<literal_value>*' specifies a scalar value. Enclose the value in quotes. For example:

```
WITH $TZ AS 'GMT'
```

An <*expression*> specifies a piece of HawkEye AP SQL code. Do not enclose the expression in quotes. For example:

```
WITH $YEAR AS _timeformat("%Y",_now(),$TZ)
```

The expression can span text lines. It ends when the next SQL keyword is encountered.

Overriding Macro Values

Generally you treat macros as programming constants. Their literal values do not change later within the SQL code block. However, the values of macros defined with literals *can* be overridden before a load operation begins.

For example, the following macro in a PTL file is designed to be overridden:

```
WITH $TZ AS 'GMT'
```

When a PTL file loads log entries, their timestamps must be converted from local time to GMT time. Often log entries have timestamps that lack a designated time zone. However, the time zone can be deduced from the location where a log file was generated.

For example, log files generated in San Francisco have ‘US/Pacific’ as their time zone, and log files generated in Paris have ‘Europe/Paris’ as their time zone. The $TZ macro can be overridden with correct local time zones for log files generated in San Francisco and in Paris. The following command modifies the example in “Table Names”, on page 166 to override the time zone so that it refers to 'US/Pacific':

```
atload --override=TZ:'US/Pacific' "host01:1680,host02:1680,host03:1680" \
  my_space.my_table my_ptl_file.ptl my_log.log
```

Using Mandatory Macros

Generally, you use macros to simplify the expressions you code in the SELECT statement.

PTL files fail certification if they do not declare and use the following macros:

```
WITH $TZ AS 'GMT'
WITH $YEAR AS _timeformat("%Y",_now(),$TZ)
WITH $STORE_RAW_PARSE_SUCCESS AS 'Y'
WITH $STORE_RAW_PARSE_FAILURE AS 'Y'
WITH $ADAPTER_VER AS '1.0'
WITH $APP_VENDOR AS '???'
WITH $APP_NAME AS '???'
WITH $APP_VERSION AS '???'
```

The macros have the following meanings.

Macro Name	Meaning	Intended for Override
`$TZ`	Specifies the local time zone for log-entry timestamps that do not carry the time zone.	
`$YEAR`	Code that calculates the current year for log entry timestamps that do not carry the year.	
`$STORE_RAW_PARSE_SUCCESS`	Processing flag that controls whether raw log entries are loaded if they were recognized and parsed successfully. Allowed values are 'Y' and 'N'. The value must by 'Y'.	
`$STORE_RAW_PARSE_FAILURE`	Processing flag that controls whether raw log entries are loaded if they were not recognized and could not be parsed successfully. Allowed values are 'Y' and 'N'. The value must by 'Y'.	
`$ADAPTER_VER`	Specifies the version number of the PTL file. The value here should match the value for version number in the initial comment block.	
`$APP_VENDOR`	Name of the vendor of the logging application that produces log entries that the PTL file is designed to parse and load.	
`$APP_NAME`	Name of the logging application that produces log entries that the PTL file is designed to parse and load.	
`$APP_VERSION`	Version number of the logging application that produces log entries that the PTL file is designed to parse and load.	

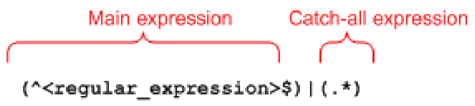
STORING RAW LOG ENTRIES

Previously in this chapter you learned the basic anatomy of a PTL file, how to declare the regular expression, the field list, and the SQL code block. You also learned how to use macros. Now you can apply that knowledge to write a basic PTL file that loads raw log entries without parse failures.

Remember that storing raw log entries is important to establishing the chain of custody, and that PTL files fail certification if they do not store all raw log entries. The keys to storing raw log entries are the use of the catch-all regular expression (`*.`) and the `s_rawmsg` or `f_rawmsg` match fields.

The Catch-All Regular Expression

The regular expression must have the catch-all regular expression. It is an alternate to the main expression you write to recognize log entries and parse information from them successfully. Your first line looks like the following example:



You can type in *`<regular_expression>`* as your regular expression for now. It is a valid regular expression. However, it will only find a match with log entries that have `"<regular_expression>"` as the entire line of text.

Match Fields for Raw Log Entries

Line 1 of your PTL file defines two memory matches. Therefore Line 2 declares two corresponding match fields: `s_rawmsg` and `f_rawmsg`. Putting Lines 1 and 2 together yields the beginning of a basic PTL file that looks like the following:

```
(^<regular_expression>$)|(.*)
s_rawmsg:VARCHAR,f_rawmsg:VARCHAR
```

The entire raw log line is either in `s_rawmsg` or `f_rawmsg` after the regular expression evaluates it for matches.

Table Columns for Raw Log Entries

The mandatory columns `AUDIT_PARSE_SUCCESS` and `unparsed_message` in the target schema are related to raw message parsing.

Column for Parse Success or Failure

The type for `AUDIT_PARSE_SUCCESS` is INT64. Two values have meaning in this column. The value 1 means that the main expression matched the raw log entry. The value -1 means that the main expression did not match the log entry and was caught by the catch-all expression.

The following code demonstrates how to set the correct value:

```
CASE
     -- a value in f_rawmsg means that main parsing failed
     WHEN f_rawmsg <> “” THEN _int64(-1)
     ELSE _int64(1)
END                     AS AUDIT_PARSE_SUCCESS,
```

One of the match fields has the raw unparsed log entry, and the other one has an empty string. The statement uses `f_rawmsg` in the conditional expression to determine which value to load in the `AUDIT_PARSE_SUCCESS` column.

Column for the Unparsed Log Entry

The type for `unparsed_message` is VARCHAR. The unparsed log entry is in one or the other match field. The macros `$STORE_RAW_PARSE_SUCCESS` and `$STORE_RAW_PARSE_FAILURE` complicate the logic.

The following code demonstrates how to store the unparsed log entry:

```
CASE
     -- store a successfully parsed raw log entry
     WHEN s_rawmsg <> "" AND $STORE_RAW_PARSE_SUCCESS=="Y"
     THEN _varchar(s_rawmsg)
     -- store an unparsed raw message raw log entry
     WHEN f_rawmsg <> "" AND $STORE_RAW_PARSE_FAILURE=="Y"
     THEN _varchar(f_rawmsg)
     -- store the empty string if storing the raw message is overridden
     ELSE _varchar("")
END                     AS unparsed_message,
```

The unparsed log entry in `s_rawmsg` or `f_rawmsg`, but the macros may be overridden with `'N'` to prevent either successful or failed log entries from being loaded. The ELSE clause ensures that the `unparsed_message` column is populated with a VARCHAR value regardless of overrides.

Varchar to Inet Conversion Failure

Two special values arise from converting varchar strings to the inet datatype. Assuming, that you are *not* executing these functions in ANSI mode, the following values are supplied:

- When no inet value exists, (meaning an empty value was provided) the column value is populated with the value **none**.
- When an inet value is not an IP address, the column value is populated with the value **invalid**.

NOTE: In the case of using an ANSI mode setting (when issuing an API or a HawkEye AP command that supports it such as **atquery**), when the above inet values, **none** or **invalid** are stored in a table, they are mapped to "**0.0.0.0/0"** when the column is read from the table**.**

Timestamps for Unmatched Log Entries

The timestamp in a log entry is not available if the main expression fails to match the entry. Yet a timestamp value is required for the ts column. The following code demonstrates how to supply the timestamp for log entries that did not parse successfully:

```
CASE
    WHEN f_rawmsg <> '' THEN _timestamp(now())
```

The `now()` function obtains the current time from the local system. The `_timestamp()` functions casts the value returned from `now()` as a TIMESTAMP.

For log entries that parse successfully, you have match fields with part or all of the log-entry timestamp. Generally, you code the ELSE clause with an expression that combines parts of the timestamp with the `$TZ` and `$YEAR` macros. The following code demonstrates the complete CASE statement for setting the value in the `ts` column:

```
CASE
```

```
     WHEN f_rawmsg <> "" THEN _timestamp(now()) -- failed timestamps
     ELSE _timestamp(now() -- successful timestamps
END AS ts,
```

TROUBLESHOOTING

These are some common problems encountered with PTL files:

- "Match Failures", next
- "Field Miscounts", on page 171
- "Incorrectly Parsed Timestamps", on page 171
- "Missing Time Zones", on page 172
- "Missing Timestamps", on page 172
- "Old Data Does Not Load", on page 172

Match Failures

The loading operation rejects source log entries when the regular expression fails to match them or when the data types specified for matching fields do not match the parsed data. For example, when a capture group matches the characters "`abc`", the value cannot be parsed into a numeric field.

Most log files have some malformed data that cannot be parsed. If the parse failure rate exceeds 1%, the problem is more likely an insufficiently flexible regular expression than it is malformed data in the source log file. To investigate the problem further, turn on the `--matchfailures` option for `atload`, and capture the rejects in a file for either retrying or debugging. If you're getting many rejects of what look like valid log entries, try testing you PTL file with fewer log entries and fewer parsed columns until you can isolate the problem.

For more information on the `--matchfailures` option, see Chapter 3: Loading, Querying, and Managing the EDW in the *Administration Guide*.

NOTE: If you include the alternative match in regular expressions, as shown under "Completing the Entire Regular Expression", on page 160, you avoid match failures completely.

Field Miscounts

For performance, the loading operation checks quickly that the number of capture groups in the regular expression matches the number of fields declared in the second line of the PTL file. Enclosing a series of capture groups within an outer set of grouping parentheses results in an extra capture group, which requires a corresponding field on the second line to avoid a miscount error. You can write your PTL file to ignore this extra field at load time by omitting it from the list of target columns in the SELECT statement.

Incorrectly Parsed Timestamps

It is important to double-check your parsed timestamp values to ensure that the regular expression is capturing them as you expected. Two simple techniques are:

- Run "`atview ... tables`" on your table after loading to get the earliest and latest timestamps, and ensure that they are reasonable.

- Create a histogram of records for each hourly time frame. For example:

  ```
  SELECT _timeformat('%D %H:00', ts) as hour, count(*)
    FROM mytable
    GROUP BY 1
    DURING ALL
  ;
  ```

Missing Time Zones

One common issue is that the timestamp field records the local time, without an explicit time-zone indicator. The log source assumes that the time zone of the system that generates the log file can be inferred. For correlation to work properly, the actual time zone must be included in the timestamp when it is loaded into the EDW. The EDW stores timestamps as GMT when the time zone is missing.

One solution is to declare WITH TIMEZONE at the top of the SQL statement, as the next example shows:

```
...
WITH TIMEZONE 'US/Pacific'
SELECT _timeparse(ts, "%Y-%m-%d %H:%M:%S", $TIMEZONE) as ts,
       field1
  FROM stdin
;
```

Missing Timestamps

Some log entries lack timestamps completely. In such cases, populate the `ts` column with the current system timestamp, as the next example shows:

```
...
SELECT now() as ts, ...
...
```

Old Data Does Not Load

Sometimes a log entry fails to load because its timestamp is earlier than the default minimum, which is Jan 1 00:00:00 2000 GMT (represented in the system as `2000-01-01T00:00:00Z`). In this case, the error message from `atload` looks like the following:

```
ts 946684799000000 not in [946684800000000,2147483647000000] for record starting
with: 2000-01-01 08:59:59,10,Kumar,Vijay,MyCo,03-5908-8194,14187,Successful
```

The loader rejects the log line on the assumption that a timestamp earlier than the default minimum indicates a user error. If your data is earlier than the default minimum, you must change the minimum value. You can make this change in one of two ways:

- Edit the `athttpd.conf` file to include the `mintimestamp` variable and set the variable to an appropriate value.

  The `athttpd.conf` file is located in:

  `<`*`Sensage_Home`*`>/etc/sls/instance/<`*`instance_name`*`>`

At the bottom of this file, add the following lines:

```
# minimum timestamp -- This value defaults to 2000-01-01T00:00:00Z
mintimestamp=<acceptable_value_in_format_yyyy-mm-ddTHH:MM:SSZ>
```

OR

- Include the following WITH statement in the SELECT statement of your PTL:

```
WITH mintimestamp <acceptable_value_in_format_yyyy-mm-ddTHH:MM:SSZ>
```

IMPORTANT: The earliest timestamp that HawkEye AP accepts is `1970-01-01T00:00:00Z`. When a PTL fails to parse the timestamp field, the loader converts the timestamp of the log line to this earliest value. Unless your data has timestamps that date to the first minute of January, 1970, you should set `mintimestamp` to a more recent value.

NOTE: The `maxtimestamp` setting does not have a default value.

INDEX