Novell® Sentinel™

6.0

www.novell.com

Volume III – SENTINEL COLLECTOR SCRIPT USER'S GUIDE

May 23, 2007

Novell®

Legal Notices

Novell, Inc. makes no representations or warranties with respect to the contents or use of this documentation, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.

Further, Novell, Inc. makes no representations or warranties with respect to any software, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to make changes to any and all parts of Novell software, at any time, without any obligation to notify any person or entity of such changes.

Any products or technical information provided under this Agreement may be subject to U.S. export controls and the trade laws of other countries. You agree to comply with all export control regulations and to obtain any required licenses or classification to export, re-export, or import deliverables. You agree not to export or re-export to entities on the current U.S. export exclusion lists or to any embargoed or terrorist countries as specified in the U.S. export laws. You agree to not use deliverables for prohibited nuclear, missile, or chemical biological weaponry end uses. Please refer to www.novell.com/info/exports/ for more information on exporting Novell software. Novell assumes no responsibility for your failure to obtain any necessary export approvals.

Novell, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at http://www.novell.com/company/legal/patents/ and one or more additional patents or pending patent applications in the U.S. and in other countries.

Novell, Inc.
404 Wyman Street, Suite 500
Waltham, MA 02451
U.S.A.
www.novell.com

Online Documentation: To access the online documentation for this and other Novell products and to get updates, see www.novell.com/documentation.

Novell Trademarks

For Novell trademarks, see the Novell Trademark and Service Mark list (http://www.novell.com/company/legal/trademarks/tmlist.html).

Third-Party Materials

All third-party trademarks are the property of their respective owners.

Third Party Legal Notices

This product may include the following open source programs that are available under the LGPL license. The text for this license can be found in the Licenses directory.

- edtFTPj-1.2.3 is licensed under the Lesser GNU Public License. For more information, disclaimers and restrictions see http://www.enterprisedt.com/products/edtftpj/purchase.html.
- Esper. Copyright © 2005-2006, Codehaus.
- jTDS-1.2.jar is licensed under the Lesser GNU Public License. For more information, disclaimers and restrictions see http://jtds.sourceforge.net/.
- MDateSelector. Copyright © 2005, Martin Newstead, licensed under the Lesser General Public License. For more information, disclaimers and restrictions see http://web.ukonline.co.uk/mseries.
- Enhydra Shark, licensed under the Lesser General Public License available at: http://shark.objectweb.org/license.html.
- Tagish Java Authentication and Authorization Service Modules, licensed under the Lesser General Public License. For more information, disclaimers and restrictions see http://free.tagish.net/jaas/index.jsp.

This product may include software developed by The Apache Software Foundation (http://www.apache.org/) and licensed under the Apache License, Version 2.0 (the "License"); the text for this license can be found in the Licenses directory or at http://www.apache.org/licenses/LICENSE-2.0. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

The applicable open source programs are listed below.

- Apache Axis and Apache Tomcat, Copyright © 1999 to 2005, Apache Software Foundation. For more information, disclaimers and restrictions, see http://www.apache.org/licenses/
- Apache Lucene, Copyright © 1999 to 2005, Apache Software Foundation. For more information, disclaimers and restrictions, see http://www.apache.org/licenses/
- Skin Look and Feel (SkinLF). Copyright © 2000-2006 L2FProd.com. Licensed under the Apache Software License. For more information, disclaimers and restrictions see https://skinlf.dev.java.net/.
- Xalan and Xerces, both of which are licensed by the Apache Software Foundation Copyright © 1999-2004. For more information, disclaimers and restrictions see http://xml.apache.org/dist/LICENSE.txt.

This product may include the following open source programs that are available under the Java license.

- JavaBeans Activation Framework (JAF). Copyright © Sun Microsystems, Inc. For more information, disclaimers and restrictions see http://www.java.sun.com/products/javabeans/glasgow/jaf.html and click download > license
- Java 2 Platform, Standard Edition. Copyright © Sun Microsystems, Inc. For more information, disclaimers and restrictions see http://java.sun.com/j2se/1.5.0/docs/relnotes/SMICopyright.html
- JavaMail. Copyright © Sun Microsystems, Inc. For more information, disclaimers and restrictions see http://www.java.sun.com/products/javamail/downloads/index.html and click download > license.

This product may also include the following open source programs.

- ANTLR. For more information, disclaimers and restrictions, see http://www.antlr.org
- Boost. Copyright © 1999, Boost.org.

- Concurrent, utility package. Copyright © Doug Lea. Used without CopyOnWriteArrayList and ConcurrentReaderHashMap classes
- Java Ace, by Douglas C. Schmidt and his research group at Washington University. Copyright © 1993-2005. For more information, disclaimers and restrictions see http://www.cs.wustl.edu/~schmidt/ACE-copying.html and http://www.cs.wustl.edu/~pjain/java/ace/JACE-copying.html
- JLDAP. Copyright 1998-2005 The OpenLDAP Foundation. All rights reserved. Portions Copyright © 1999 - 2003 Novell, Inc. All Rights Reserved.
- OpenSSL, by the OpenSSL Project. Copyright © 1998-2004. For more information, disclaimers and restrictions, see http://www.openssl.org.
- Tao (with ACE wrappers) by Douglas C. Schmidt and his research group at Washington University, University of California, Irvine and Vanderbilt University. Copyright © 1993-2005. For more information, disclaimers and restrictions see http://www.cs.wustl.edu/~schmidt/ACE-copying.html and http://www.cs.wustl.edu/~pjain/java/ace/JACE-copying.html
- Tinyxml. For more information, disclaimers and restrictions see http://grinninglizard.com/tinyxmldocs/index.html.
- Java Service Wrapper. Portions copyrighted as follows: Copyright © 1999, 2004 Tanuki Software and Copyright © 2001 Silver Egg Technology. For more information, disclaimers and restrictions, see http://wrapper.tanukisoftware.org/doc/english/license.html.

NOTE: As of the publication of this documentation, the above links were active. In the event you find that any of the above links are broken or the linked web pages are inactive, please contact Novell, Inc., 404 Wyman Street, Suite 500, Waltham, MA 02451 U.S.A.

Preface

The Sentinel Technical documentation is general-purpose operation and reference guide. This documentation is intended for Information Security Professionals. The text in this documentation is designed to serve as a source of reference about Sentinel's Enterprise Security Management System. There is additional documentation available on the Sentinel web portal.

Sentinel Technical documentation is broken down into five different volumes. They are:

- Volume I – Sentinel™ 5 Install Guide
- Volume II – Sentinel™ 5 User's Guide
- Volume III – Sentinel™ 5 Collector User's Guide
- Volume IV – Sentinel™ 5 User's Reference Guide
- Volume V – Sentinel™ 3rd Party Integration
- Volume VI – Sentinel™ Patch Installation Guide

Volume I – Sentinel Install Guide

This guide explains how to install:

- Sentinel Server
- Sentinel Console
- Sentinel Correlation Engine
- Sentinel Crystal Reports
- Collector Builder
- Collector Manager
- Advisor

Volume II – Sentinel User's Guide

This guide discusses:

- Sentinel Console Operation
- Sentinel Features
- Sentinel Architecture
- Sentinel Communication
- Shutdown/Startup of Sentinel
- Vulnerability assessment
- Event monitoring
- Event filtering
- Event correlation
- Sentinel Data Manager
- Event Configuration for Business Relevance
- Mapping Service
- Historical reporting
- Collector Host Management
- Incidents
- Cases
- User management
- Workflow
- Collector Host Management
- Collector Manager

Volume III – Collector User's Guide

This guide discusses:

- Collector Builder Operation
- Collectors
- Building and maintaining Collectors

Volume IV - Sentinel User's Reference Guide

This guide discusses:

- Collector scripting language
- Collector parsing commands
- Collector administrator functions
- Collector and Sentinel meta-tags
- Sentinel correlation engine
- User Permissions
- Correlation command line options
- Sentinel database schema

Volume V - Sentinel 3rd Party Integration Guide

- Remedy
- HP OpenView Operations
- HP Service Desk

Volume VI - Sentinel Patch Installation Guide

- Patching from Sentinel 4.x to 6.0
- Patching from Sentinel 5.1.3 to 6.0

Contents

1 Introduction **1-2**

Contents 1-2
Conventions Used 1-2
Notes and Cautions 1-2
Commands 1-2
Collector Scripts 1-2
Other Sentinel References 1-3
Contacting Novell 1-3

2 Collector Scripts **2-2**

Out-of-the-Box Collector Scripts 2-2
Collector Script Structure 2-3
Template Files 2-3
Startup and Backout Chain Files 2-6
Parameter Files 2-6
Lookup Files 2-7
Mapping Files 2-7
Manifest Files 2-8
Package File 2-8
Collector Directory Structure 2-8

3 Developing and Maintaining Collector Scripts **3-2**

Collector Script Development Overview 3-2
Building a Collector Script 3-3
Creating and Configuring Template Files 3-4
Creating and Configuring Parameter Files 3-8
Sentinel 6 Compatible Parameter Conventions 3-9
Creating and Configuring Lookup Files 3-9
Scripts 3-10

1 Introduction

The Collector Script User's Guide explains what a Collector Script is and how to create or modify one using the Sentinel Collector Builder.

This guide assumes that you are familiar with Network Security, Database Administration, Windows and UNIX operating systems.

For more information on configuring, debugging and deploying collectors, see Event Source Management in *Sentinel 6.0 User's Guide*.

Contents

This guide contains the following chapters:

- **Chapter 1:** Introduction
- **Chapter 2:** Collector Scripts
- **Chapter 3:** Developing and Maintaining Collector Scripts

Conventions Used

Notes and Cautions

NOTE: Notes provide additional information that may be useful.

CAUTION: Cautions provide additional information that may keep you from performing damage or loss of data to your system.

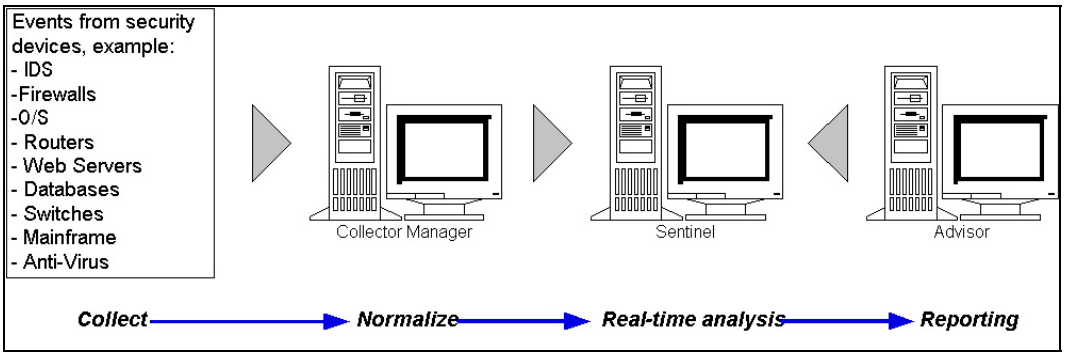
Commands

Commands appear in courier font. For example:

```
useradd –g dba –d /export/home/oracle –m –s /bin/csh
   oracle
```

Collector Scripts

Collector Scripts are plug-ins that enables Sentinel to normalize raw data into a format that is understood by other Sentinel components. For example, a Collector Script can be written to parse log data from a particular IDS product and convert the data into a Sentinel event. The Sentinel event can then be visualized in Active Views, processed by Correlation Engine, queried in a report, and added to an incident response workflow. Collector Scripts can also parse non-event data, such as vulnerability scan data. In this case, the Collector Script transforms the raw scan data into a format understood by Sentinel. Sentinel can then store the vulnerability data in the database and include it in the Exploit Detection map.



Novell has already developed many Collector Scripts that support parsing data from many of the most popular devices. These Collector Scripts can be used without modification and allow you to easily get started collecting critical data from your important devices. You can download Collectors from Novell Collector Download page (http://support.novell.com/products/sentinel/collectors.html)

For more information about Collector Scripts, see Chapter 2, “Collector Scripts”.

Other Sentinel References

The following manuals are available with the Sentinel install CDs.

- Sentinel™ Install Guide
- Sentinel™ User’s Guide
- Sentinel™ Collector Builder User's Guide
- Sentinel™ User's Reference Guide
- Sentinel™ 3rd Party Integration Guide
- Release Notes

Contacting Novell

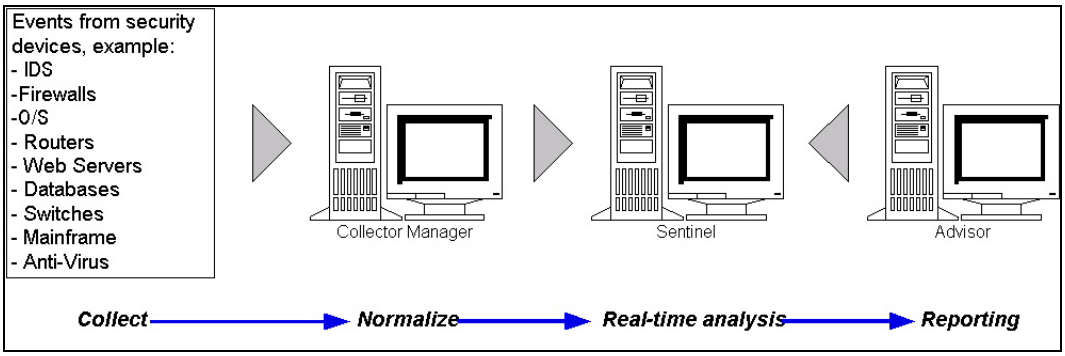
- Website: http://www.novell.com
- Novell Technical Support: http://support.novell.com/phone.html?sourceidint=suplnav4_phonesup
- Self Support: http://support.novell.com/support_options.html?sourceidint=suplnav_supportprog
- Patch Download Site: http://download.novell.com/index.jsp
- 24x7 support: http://www.novell.com/company/contact.html

2 Collector Scripts

Collector Scripts are capable of parsing and normalizing data from many sources, including:

- Intrusion Detection Systems (host)
- Intrusion Detection Systems (network)
- Firewalls
- Operating Systems
- Policy Monitoring
- Authentication
- Routers & Switches
- VPN
- Anti-Virus
- Web Servers
- Databases
- Mainframe
- Vulnerability Assessment
- Directory Services
- Network Management
- Proprietary Systems

Collector Scripts are used by deploying them in Event Source Management in the Sentinel Control Center. When adding a Collector node to Event Source Management, it will prompt for the Collector Script you wish to use to parse the data from the associated Event Sources. Choose the Collector Script that is appropriate for the data you expect to receive from the Event Source. For more general information on Event Source Management, see Event Source Management in the *Sentinel 6.0 User's Guide*. For more information about specific Collector Scripts, see the documentation included with the Collector download.



Out-of-the-Box Collector Scripts

Novell has already developed many Collector Scripts that support parsing data from many of the most popular devices. These Collector Scripts can be used without modification in most environments and allow you to easily get started collecting critical data from your important devices. You can download Collectors from Novell Collector Download page (http://support.novell.com/products/sentinel/collectors.html).

From a Novell support perspective, there are two categories of Collector Scripts. They are:

- **Supported Collector Scripts:** Collector Scripts in this category have all of the following attributes:
  - Full Documentation
  - Meta-data: This includes data such as translation maps and information that Sentinel Control Center uses to make deployment and configuration easier.

- QA/UAT certified
- General availability to all customers
- Support from Novell Technical Support

- **Unsupported Collector Scripts:** Collector Scripts in this category may have one or more of the following attributes:
  - A result of a proof of concept
  - Developed for a specific customer
  - Minimal documentation (or none)
  - Missing meta-data
  - Not supported by Novell Technical Support

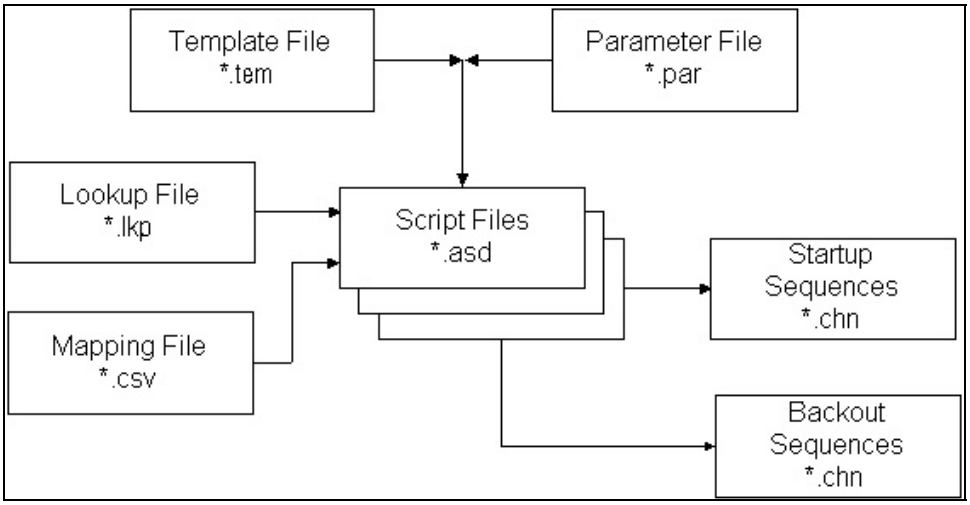
NOTE: All of the Collector Scripts available on the Novell Collector website are fully supported and have all the features described in the *Supported Collector Scripts* category above.

Collector Script Structure

Collector Scripts are made of the following files:

- "Template"
- "Startup and Backout Chain"
- "Parameter"
- "Lookup"
- "Mapping"
- "Manifest"
- "Package"

These files work in conjunction to provide the parsing, configuration, and event enrichment capabilities found in Collector Scripts.



Template Files

You can create, add states, edit and delete templates. Templates determine the high level flow of a collector's processing. Most of the decisions about templates revolve around what types of records you are working with and their format. Each collector contains at least one template file with a .tem extension. When developing a Collector Script using Collector Builder, template files are located in the directory:

```
%ESEC_HOME%\data\collector_workspace\<Collector Name>
```

Each template file is named after the column name of the set of values in the parameter file. Template files are grouped in an ordered sequence into startup and backout sequences. For more information on startup and backout sequences, see "Startup and Backout Chain Files".

Template files contain a set of states. A state is a point of action within the logical flow or path of a template. Each state contains information indicating the action to perform. States also may include references to parameters. When the Collector Script template is built into an *.asd* file, parameter references in template states are resolved. The resulting *.asd* file contains the parameter values instead of the parameter reference.

A template can contain the following types of states:

- "Transmit (Tx)"
- "Receive (Rx)"
- "Decide"
- "Parse"
- "Next"
- "Go"
- "Stop"

When a state is inserted into a template, it is assigned a number. The number is assigned using the following logic.

- The *Transmit*, *Receive*, *Decide* and *Parse* states are numbered in the order that they are inserted in the template. The original number assigned to these states does not change, even if the states are reordered in the template.
- The *Next* and *Go* states are identified by the number of the state to which they are pointing.
- The *Stop* state is always assigned the number zero.

Transmit State

Transmits data to a Connector; The Transmit State sends either a string or variable (depending on what type of data is selected) to the Connector associated with the Collector.

Receive State

Reads data from a Connector; The Receive State specifies the method used to determine when data has been received from the Connector. In the Receive State, you specify:

- Receive Type
- Minimum Bytes
- Delimiter Decide String

After the Receive State of the RxBuffer, the Connector, by default, will populate at least two variables automatically with the results of the Receive State:

- s_RXBufferString contains the text received by the RxBuffer
- i_RXBufferLength contains the length of the s_RXBufferString

This is equivalent to executing the following script code after a Receive State:

- COPY (s_RXBufferString)
- LENGTH(i_RXBufferLength,s_RXBufferString)

These automatically populated variables allow for easy comparison in a Decide State of whether or not the Receive State timed out (i_RXBufferLength = 0). They also allow for the direct use of the RXBuffer through the s_RXBufferString variable. This behavior can be modified in the connector and is implemented for backward compatibility.

Receive Types: There are four Receive Types available in the Template editor. They are:

- **Timeout:** Allows a script to continue processing even if data is not received in a specified amount of time. Selecting timeout allows the receiving of data until the timeout period is reached, as defined by the variable, RX_TIMEOUT_DELAY.

  After the Receive State is entered, processing stops until the minimum bytes is read or RX_TIMEOUT_DELAY seconds passes. After receiving more than the minimum number of bytes specified or when the timeout has been exceeded, the Collector processing continues to the next state of the script. If the minimum number of bytes is received, the data is placed in the Rx_Buffer.
- **Wait**: Used primarily when receiving unsolicited event messages. The Collector Manager will wait for the "timeout" duration until data is received.

  After the Receive State is entered, processing stops until the minimum number of bytes specified in the Minimum Bytes box are received. After more than the minimum number of bytes specified is received than in the Minimum Bytes box, the data is placed in the Rx_Buffer and the Collector processing continues to the next state of the script. If the minimum number of bytes is not received, the Collector processing never times out.
- **Delim timeout**: Uses a pre-defined string of characters to indicate that data has been received. The data in the Delimiter Decide String box is verified against the data in the receive buffer as each byte is received.

  If the delimiter decide string (such as an end-of-line sequence) is encountered after the minimum byte position set in the Minimum Bytes box is received, the data up to and including the delimiter is stored in the Rx Buffer. If the delimiter decide string is not encountered, no data is transferred to the receive buffer and the Collector processing times out in the default timeout period.
- **Delim wait**: Used when waiting for unsolicited messages. A user-defined string of characters that indicates that data has been received. The data is used in the Delimiter Decide String box to verify the receive data as each byte is received. The parameter RX_TIMEOUT_DELAY has no effect when using the delim wait option.

  If the delimiter decide string is encountered after the minimum number of bytes set in the Minimum Bytes box is received, the Collector processing continues and the data is placed in the Rx_Buffer. If the delimiter decide string is not encountered, no data is transferred to the receive buffer and the script does not timeout. The Collector processing never times out. In addition, if the delimiter decide string is encountered, but the minimum bytes have not been received, the Collector processing never times out.

Minimum Bytes: The minimum number of bytes is the number of bytes that must be received before the default timeout period used or continues processing. Processing in the script will not continue until the minimum number of bytes is received.

Delimiter Decide String: The Delimiter Decide String is completed when the Receive Type is delim timeout or delim wait. Collector processing will not continue to the next state until the delimiter decide string matches data read in and the minimum number of bytes has been received.

The delimiter decide string is a POSIX 1003.2 compliant regular expression.

Decide State

Determines which state to go to next based on an input value; The Decide State evaluates the contents of the receive buffer or a variable to determine what action to take. If the information in the receive buffer satisfies the specified criteria, the flow continues down

the *Yes* path. If the receive buffer does not satisfy the specified criteria, the flow continues down the *No* path.

There are four Decide Types. They are:

- **String:** Compares a user-defined decide string to the content of the receive buffer. The contents of the decide string are verified with the contents of the receive buffer, or a variable, to determine which decision route to process. The decide string is a POSIX 1003.2 compliant regular expression. A variable supports strings, integers and floats.
- **True**: Forces an evaluation of true, Collector Manager follows the Yes route.
- **False:** Forces an evaluation of false, Collector Manager follows the No route.
- **Data:** Compares a user-defined decide string to another string or the value of a variable.

Parse State

Contains customized parsing logic to process raw data; The Parse State contains the script's parsing commands. The parsing commands in this kind of state can include references to parameter, which will automatically be resolved when the template file is built into an *.asd* file. The Collector Builder contains a Visual Editor and a Text Editor to edit the parsing command in a Parse state.

Next

Jumps to a specific state in the next template in the chain

Go

Jumps to a specific state in the current template

Stop

Stops Collector Script processing

Startup and Backout Chain Files

Each Collector Script contains a startup and backout chain file. These files define the order in which to execute the template files when a Collector is started or stopped. A template must be included in a startup or backout sequence in order to do any useful processing.

Parameter Files

Parameters enable the end-user to customize the behavior of a Collector Script without changing the Collector Script code. Parameter files (.par) are represented in Collector Builder as a table that maps parameter names and their values to template files. Parameters are stored as strings. Any numeric value needs to be converted from a string type to a number type before numeric manipulation can take place. Before Sentinel 6, when a user updated the value of a parameter, they needed to build the Collector Script for the new value to take effect. Building a Collector Script merges the parameter values with the template file to create a script file (.asd). In Sentinel 6, the Collector Manager automatically builds the script upon starting a Collector.

Template file names are displayed in the first row of the table and the parameter names or labels are displayed in the first column of the table. The second row of the table is used to define the icons that appear in the Builder's Collector tree. The remaining row defines the variables or parameter values to be used for parameter as it relates to the particular template.

Lookup Files

Lookup files are optional Collector files (.lkp) which contain match cases. Each match case may contain parsing code to execute. Based on the match cases in a specific lookup file and the data received from event sources, the LOOKUP parsing command will determine whether the named match case is found or is not found. Additionally if there is associated parsing code, it will be executed.

The standard Collector Script template, available for download from the Novell Collector website (http://support.novell.com/products/sentinel/collectors.html), contains the following lookup files:

- **Template:** Contains parsing code used by the template file. This lookup file should not be modified.
- **Core:** Contains standard parsing code specific to the device supported by the Collector Script. This is the lookup file where you should put parsing code when you are developing a new Collector Script. This lookup file should not be modified if customizing an existing Collector Script.
- **Agent:** This lookup file is the best place to put custom parsing code when extending the functionality of an existing Collector Script. The *agent* lookup file, by default, calls into the corresponding lookup match cases in the *core* lookup file.

These lookup files are organized in such a way that the Collector Script can easily be extended to meet the needs of a specific user without the user needing to modify the standard parsing code. All user-specific parsing code should be added only to the *agent* lookup file. It is important to maintain this structure so that it is easier to debug and upgrade customized Collector Scripts in the future.

If you need to modify the behavior of an existing collector you can do so in the below mentioned lookup cases in agent.lkp.

- **Setup:** one time setup of variables and parameters (collector specific parameter validation)
- **Initialize_Vars:** the beginning of every loop, where variables are initialized once per parse (variable initialization)
- **Parse_Line:** the place where the parsing is performed

Mapping Files

Mapping files are optional files (.csv) that allow for fast lookup of key entries. A mapping file is used by calling the TRANSLATE parsing command and giving it the csv file to load. The csv file given to the TRANSLATE command is a relative path from a Collector Script's directory. The editing of these files is currently not available within Collector Builder, but the files can be edited using Excel.

Example of a possible mapping file is:

```
~Month~   ~Number~
Jan              1
Feb              2
Mar              3
Apr              4
May              5
Jun              6
Jul              7
Aug              8
Sep              9
Oct             10
```

```
Nov             11
Dec             12
```

The first column is used as the key for the mapping. Each additional column contains values that can be mapped to script variables (string, variable or float) using the TRANSLATE parsing command. This particular example is used to translate (map) Month to a Number (For example, Jan to 1).

Manifest Files

The manifest file *agent.nfo*, is used by both the Collector Manager and the standard Collector Script Template. It contains information related to the device supported by the Collector Script.

The following is the contents of the manifest file:

- agent.nfo
  - product,Snort
  - product.vendor,GNU
  - product.version,2.0
  - product.security.type,IDS
  - product.sensor.type,N
  - product.name,IDSx_GNUx_SNRT
  - file.version,1

Package File

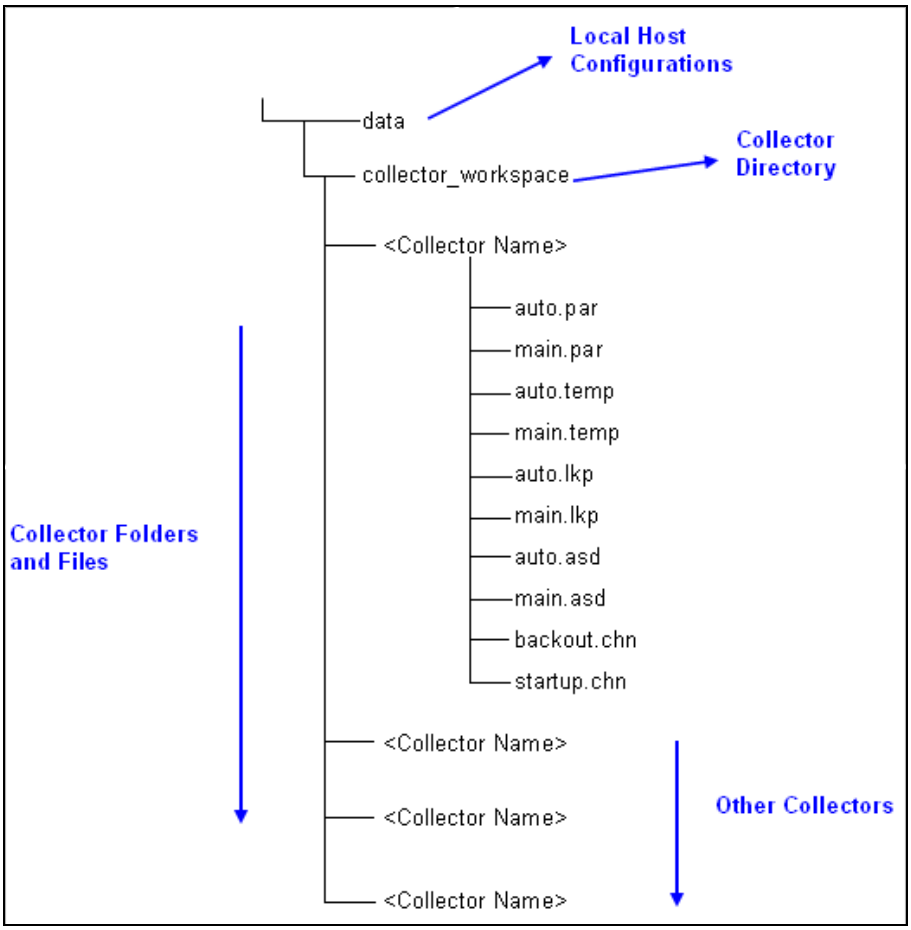
The Collector Script Package file (*package.xml*), introduced in Sentinel 6, contains meta-data describing many attributes of the Collector Script, including:

- Name
- Description
- Version
- Parameter Definitions, including name, description, default values, and options
- Supported Devices
- Supported Connectors and default Connector settings

Sentinel applications access this meta-data to customize their behavior to best handle the Collector Script. For example, Sentinel Control Center will access this meta-data to simplify the Event Source Management wizard screens.

Collector Directory Structure

The following represents the Collector Directory structure in Sentinel:

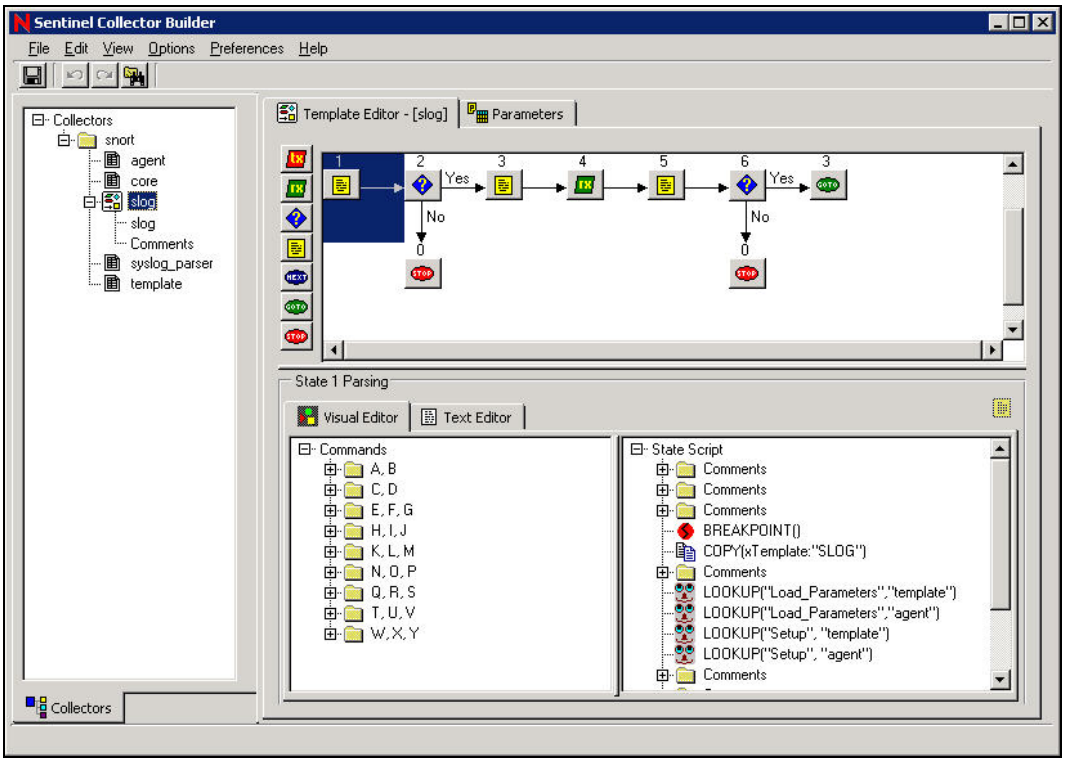


Key	Description
data	Port configuration files (Collector Hosts)
data\collector_workspace	Collector files
.par	Parameter files
.tem	Template files
.lkp	Lookup files
.asd	Active state description files
backout.chn	Backout script files
startup.chn	Startup script files

3 Developing and Maintaining Collector Scripts

Collector Scripts can be fully customized or built from scratch by any user. Sentinel provides the Collector Builder development environment for building and maintaining Collector Scripts. The Collector Builder is an integrated development environment (IDE) that makes developing Collector Scripts easy.

Since Sentinel 6, the functions of configuring, debugging and deploying Collector Scripts has been moved to the Event Source Management feature of Sentinel Control Center. For more information on these functions, see Event Source Management in *Sentinel 6.0 User's Guide*.



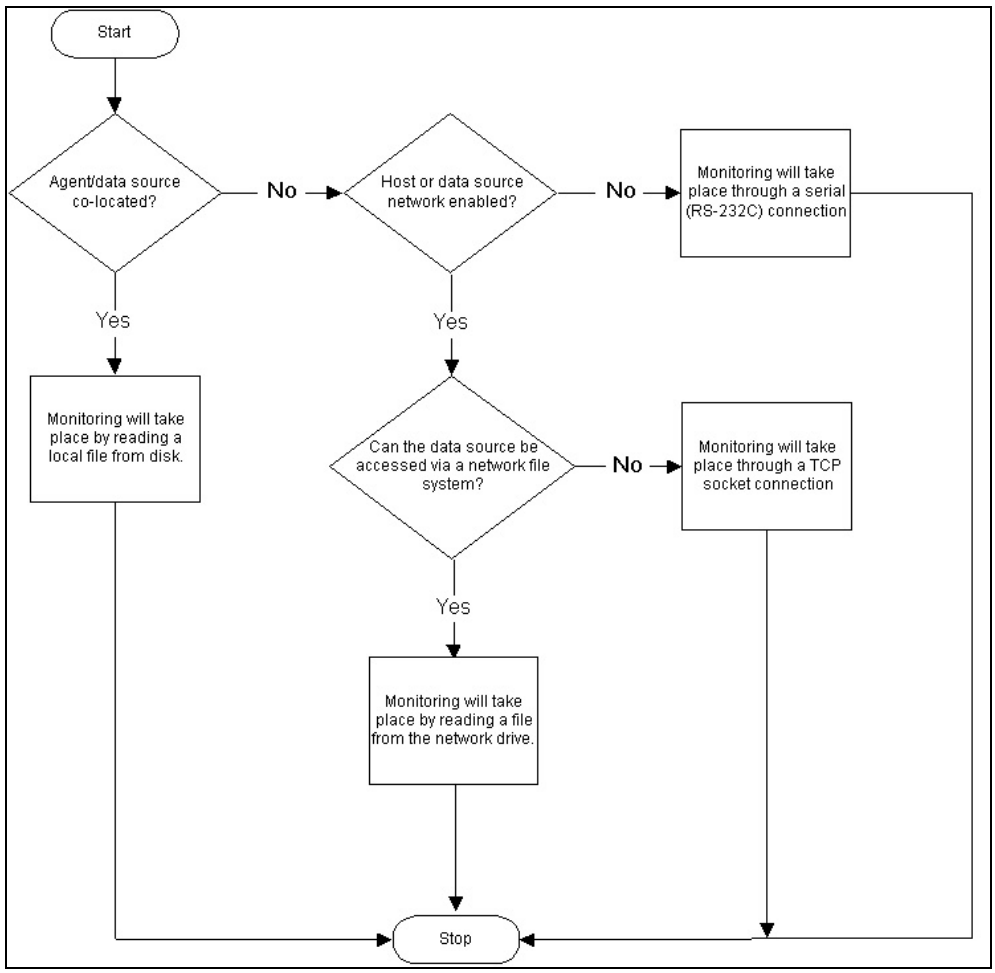
Collector Script Development Overview

The following are the basic steps in implementing a Collector Script.

1. Determine what you want to monitor
2. Determine how to monitor the data
3. Determine the product's operating system
   - If the host and product are co-located, the most logical way to obtain the data is to read it from the product's log file.

- If the host and product are not located on the same machine, the needed data can be obtained through a network file system setup (such as NFS, Samba or SMB share), a TCP/IP socket connection or a serial connection.

4. Create the Collector Script to parse the data from the data source you want to monitor. For more information, see "Building a Collector Script".
5. Import the Collector Script from the collector workspace directory and deploy the Collector Script using Event Source Management in Sentinel Control Center. For more information on these functions, see Event Source Management in *Sentinel 6.0 User's Guide*.



Building a Collector Script

Building a Collector Script requires you to create the following files:

- "Template files"
- "Parameter files"
- "Lookup files" (optional)

The most efficient way to develop a Collector Script is to do one of the following:

- Find an existing Collector Script that parses similar data that is acquired using a similar connection method as the data you want to parse.
- Download the Collector Script Template from the Novell Collector website (http://support.novell.com/products/sentinel/collectors.html)

By using one of the above options, you can start creating the parsing code specific to the device you want to collect data from without having to create template, parameter, and lookup files from scratch.

To use an existing Collector Script do one of the following, depending on where you obtained the existing Collector Script:

If you downloaded the Collector Script from the Novell Collector website:

1. Download the *Collector Installer* from Novell Collector website (http://support.novell.com/products/sentinel/collectors.html)

   NOTE: Some of the Collectors may require an installer to install. In that case, the instructions to install the Collector are explained in the specific Collector related documentation.

2. Run the *Collector Installer*, with the *installedAgentDirectory* command argument set to *%ESEC_HOME%\data\collector_workspace* and the *srcAgentDirectory* command argument set to the directory where the downloaded Collector Script zip file is located. This will place the contents of the zip file in the collector workspace directory.
3. Click *View* > *Refresh Collectors* in Collector Builder to load the newly installed Collector Scripts.

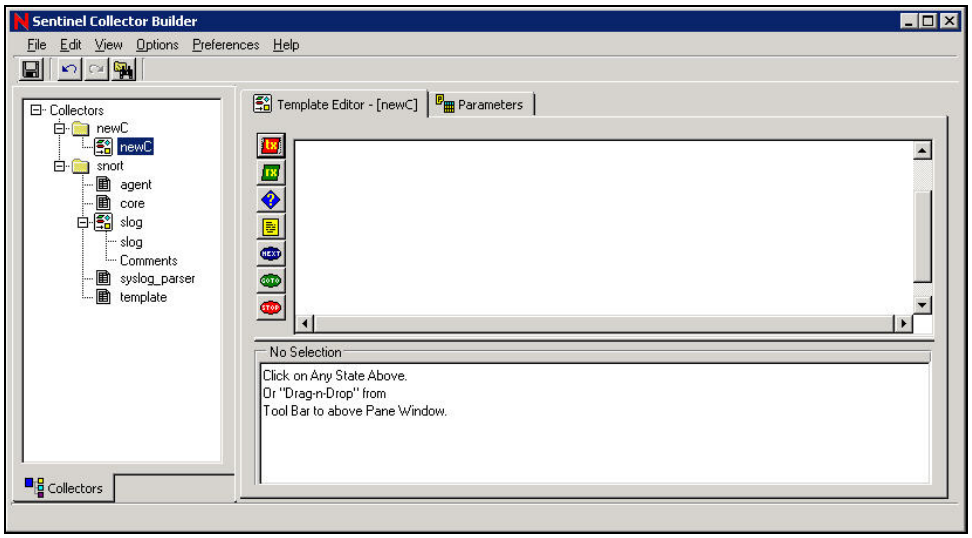
If you have the Collector Script deployed in Event Source Management:

1. Stop the Collector you wish to modify.
2. Right click on the Collector and select *Debug…*
3. In the Collector Debugger window, select the *Upload/Download* tab.
4. Ensure the destination directory field points to the collector workspace directory.
5. Click the *Download* button. The Collector Script will be downloaded from the Plug-in Repository to your local collector workspace directory.
6. Click on *View* > *Refresh Collectors* in Collector Builder to load the downloaded Collector Scripts.

Creating and Configuring Template Files

To creating and configuring Template files:

1. Start Collector Builder.
2. In the Collectors tree, right-click *Collectors* and click *New Collector*.
3. Enter the new Collector name in the space provided and press Enter.
4. Right-click on the new *Collector* and click *New Template.*



5. In the New Template box on the Collectors tree type a new template name and press Enter.
6. Select the new template and click the *Template Editor* tab.
7. In the *Template Editor* panel, drag and drop states to the editing area using the state buttons on the left of the panel. For information about adding states to a template, see "Adding a States to a Template".
8. Click *Save*.

Adding a State to a Template File

All Collectors begin processing at state 1, regardless of where state 1 appears in the template. Assuming state 1 is a processing state, insert the new state following state 1.

Collector Builder automatically assigns the first state a state number of 1. It is recommended that this first state contain only a BREAKPOINT() parsing command. Putting only a breakpoint after State 1 will allow for easier debugging. When debugging, the parser will automatically stop on the next state.

When building a template, start with a 'breakpoint only' parse state. Then, add the working state (Receive state, Parse state, and so on..) at State 2. If you need to add a state to the beginning of the template, insert it after the BREAKPOINT only.

Do not delete the BREAKPOINT only parse state unless it is necessary to add another state at the beginning of the template. Optionally, you can enter comments in this BREAKPOINT only about the functionality of the template.

To Add a State to a Template:

1. Click the *Collectors* tab to open the Collectors tree panel.
2. In the Collectors tree, select a template to display the Template Editor in the right panel.
3. Click *Options* > *Add State* > *Transmit*, *Receive*, *Decide*, *Parse*, *Next*, *Go To* or *Stop* states as needed or click the appropriate buttons.

- Transmit
- Receive
- Decide
- Next
- Go To
- Stop

-  Parse

4. Using the editing panels at the bottom of the Template Editor panel, insert the new code into each state as you add it.

   Alternatively, you can also drag and drop a Parse State button from the left side of the Template Editor into the editing area.

NOTE: Do not use double quotation marks as part of the decide string either in the receive state (to match the delimiter in a log file, for example) or in a decide state, you will get the following error message:

```
***ERROR: Reading Template File..."
```

When one or more quotes is put into the decide or delimiter string, a quote mismatch occurs as follows:

```
StateDecideString: "test"123"
```

The workaround is to use \22\ instead of a quote (").

NOTE: If you select another item in the Collectors tab (even in the same Collector) and then go back to the offending template, Collector Builder gives you this error message and will not display any part or states of the template. The error is occurring because the quote character (") is used to delimit field values in a .tem file. For example:

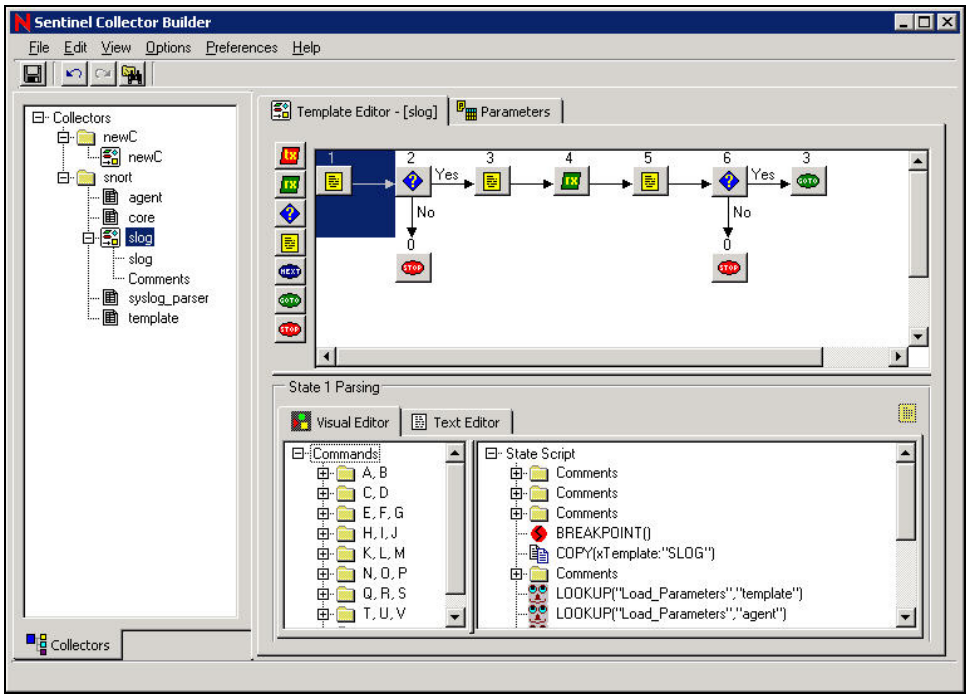
```
StateDecideString: "test"
StateDelimiterString: "123"
```

Entering a Parsing Command through the Visual Editor

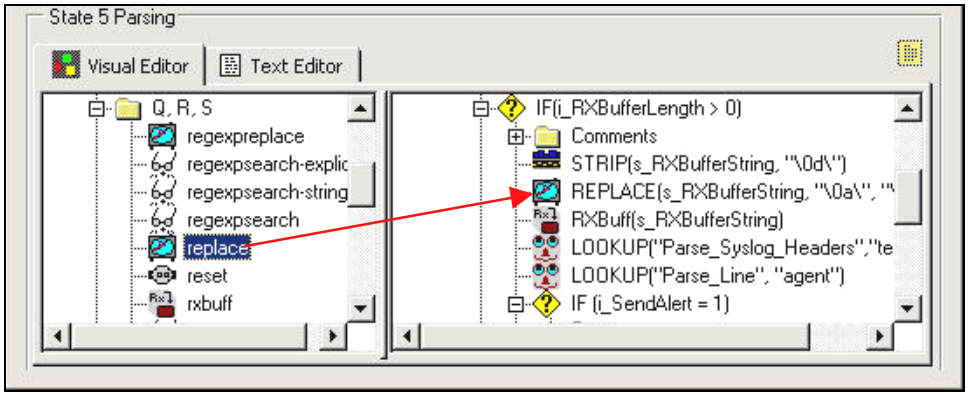
There are two methods of entering a Parsing Command, using the Visual Editor or using the Text Editor. Limit your commands to no more than 4096.

To enter a Parsing Command through the Visual Editor:

1. In the Template Editor, select a parse state. The Visual Editor tab is open by default when you click a template to open.



2. In the visual editor, drag the parsing commands to the right side of the panel.



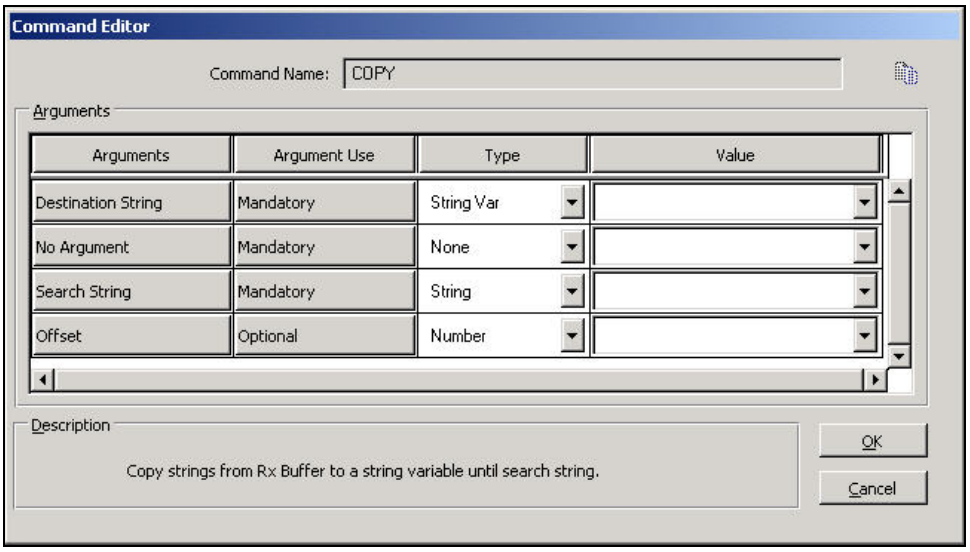
3. Enter the argument values in the Popup Command Editor window.
   - **Select a Type:** The types for each parsing command is described in the *Sentinel 6.0 User's Reference Guide..*
   - **Specify a Value:** Values are defined for a specific application. Examples of values for each parsing command are in the *Sentinel 6.0 User's Reference Guide.*

To enter a Parsing Command through the Text Editor:

1. In the Template Editor, click the *Text Editor* tab.
2. Manually enter your parsing commands.

   Use the Tab key on the keyboard to line up text when using a fixed font. Copy, cut and paste options function like any standard text editor.

Editing a Parsing Command



- **Arguments:** Includes all of the possible arguments for the parsing command you selected in the Visual Editor
- **Argument Use:** Defines whether the argument is mandatory or optional
- **Type:** Determines the variable type; for example, strings, string variables, numbers, number variables, floats, float variables or predefined variables
- **Value:** Value you define for the variable that's named in the Type column

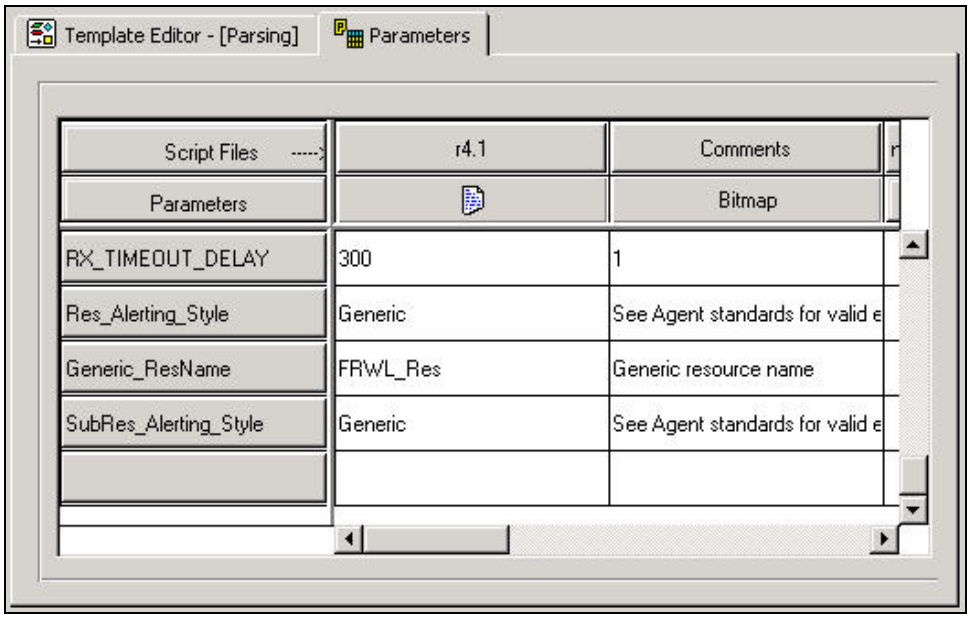
Editing a Parsing Command

1. In the visual editor, either:
   - Right click on a parsing command and choose *Add to State Script*
   - Double-click on a parsing command, the Command Editor will open
2. Fill in the Type and Value boxes to complete the editing. For more information on Parsing Command descriptions, see *Sentinel 6.0 User's Reference Guide.*

Creating and Configuring Parameter Files

To creating and configuring Parameter Files:

1. Select a template and click the *Parameters* tab in the right panel.



2. Double-click the *new...* button in the second column of the Parameters table.
3. Enter the new parameter name (this is the name of your script, such as r4.1) and press Enter.
4. (Optional) Right-click the *Bitmap button* (second column/second row) and click *Assign Bitmap*. In the Bitmap Assignment dialog box, select a *Bitmap* button.
5. Double-click each of the new parameter boxes and enter the appropriate values.
6. When all of the values are defined, the parameter and the template file need to be compiled to create a script. Go to section "Building Scripts".

Sentinel 6 Compatible Parameter Conventions

Use the following conventions when creating parameters to ensure compatibility with Sentinel 6 Event Source Management features.

- First column in the Parameters screen has the default values
- Second column has the description of the Parameters
- In a collector, if you have the same parameters in multiple scripts, the script name will precede the parameter name (internal, not display name).

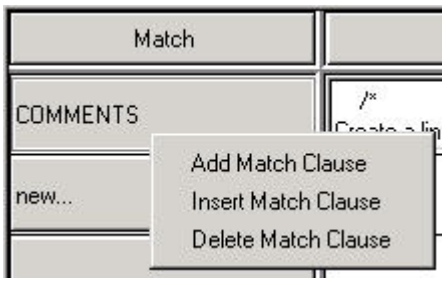
Creating and Configuring Lookup Files

This is an optional procedure.

To create and configure Lookup Files:

1. Right-click a Collector and click *New Lookup File*.
2. In the New Lookup File box, type a new lookup file name and press Enter.
3. In the Match column, Double-click *new...* and enter the string to match and press Enter. You can add, insert and delete match clauses.
   - **To add:** In the Match column, right-click a match clause and click *Add Match Clause*.
   - **To insert:** In the Match column, right-click a match clause and click *Insert Match Clause*.

- **To delete:** In the Match column, right-click a match clause and click *Delete Match Clause*.



4. (Optional) To enter parsing commands, right-click in the Parsing column to open the Visual Editor. For information about using the Visual Editor, see "To Enter Parsing Commands Using the Visual Editor".
5. Select the parsing commands and complete them in the Command Editor window. The commands display in the Parsing column.
6. When all of the values are defined, it must be compiled to create a script. Go to section "Building a Script".

Scripts

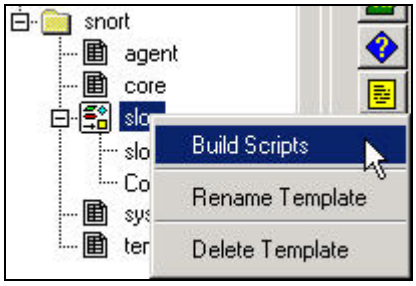
Scripts (*.asd* files) are generated from a combination of the template files and parameter files. You can generate multiple scripts from one template. Collector Builder allows you to:

- "Build a script"
- "Assigning a Startup Sequence to a Script"

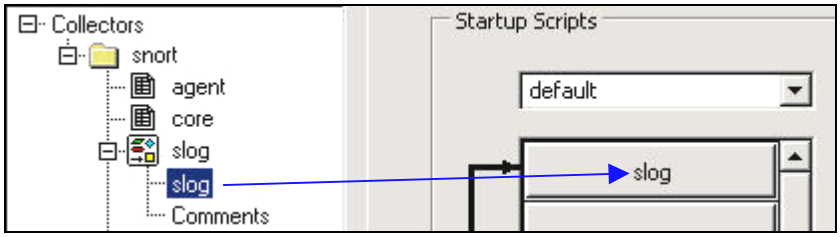
Building a Script

To Build a script:

1. In the left panel, select the template that you are building the scripts from.
2. Select *File* > *Build Scripts*.



3. In the Template Editor tab, drag a script from the template to the Startup Scripts or Backout Scripts column in the right panel.



Scripts execute in the order they appear in the Startup Scripts and Backout Scripts columns. To rearrange the script order, drag the scripts up or down in the columns.

NOTE: The final script in a backout sequence must end with the Stop processing state.

4. Click File > Save.

Assigning a Startup Sequence to a Script

If you want a port to run at startup, you can assign a startup sequence to run a specified set of scripts at startup. A startup sequence is a file that contains the names of the scripts to run at startup.

NOTE: For compatibility with Sentinel 6, the startup sequence filename must be *startup.chn*.

To assigning a Startup Sequence to a Script:

1. Right-click a script name in the Collectors tree and select New Startup Sequence. The New Startup Sequence dialog box displays.
2. In the New Startup Sequence dialog box, type the sequence name and click *OK*. The new startup sequence name is added to the menu at the top of the Startup Scripts panel. The following restrictions apply to sequence names:
   - Do not use startup or backout as sequence names
   - Do not use duplicate sequence names within the same Collector
3. Drag the script file names from the Collectors tree to the Startup Scripts column. The scripts execute in the order that they appear in the column, from top to bottom.
4. To rearrange the script order, drag the scripts from the column, or right-click the *Startup Scripts* panel and select *Reorder Startup Script.*