RSA enVision 4.1 SP1
Universal Device Support Guide

Contact Information

Go to the RSA corporate web site for regional Customer Support telephone and fax numbers**:** **www.rsa.com**

Trademarks

RSA, the RSA Logo, RSA enVision, RSA Event Explorer and EMC are either registered trademarks or trademarks of EMC Corporation in the United States and/or other countries. All other trademarks used herein are the property of their respective owners. For a list of EMC trademarks, go to **www.rsa.com/legal/trademarks_list.pdf**.

License agreement

This software and the associated documentation are proprietary and confidential to EMC, are furnished under license, and may be used and copied only in accordance with the terms of such license and with the inclusion of the copyright notice below. This software and the documentation, and any copies thereof, may not be provided or otherwise made available to any other person.

No title to or ownership of the software or documentation or any intellectual property rights thereto is hereby transferred. Any unauthorized use or reproduction of this software and the documentation may be subject to civil and/or criminal liability. This software is subject to change without notice and should not be construed as a commitment by EMC.

Third-party licenses

This product may include software developed by parties other than RSA. The text of the license agreements applicable to third-party software in this product may be viewed in the thirdpartylicenses.pdf file.

Portions of this application include technology used under license from Visual Mining, Inc. 2000-2010.

Portions of this application include iAnywhere technology, 2001-2010.

Note on encryption technologies

This product may contain encryption technology. Many countries prohibit or restrict the use, import, or export of encryption technologies, and current use, import, and export regulations should be followed when using, importing or exporting this product.

Distribution

Use, copying, and distribution of any EMC software described in this publication requires an applicable software license.

EMC believes the information in this publication is accurate as of its publication date. The information is subject to change without notice.

THE INFORMATION IN THIS PUBLICATION IS PROVIDED "AS IS." EMC CORPORATION MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WITH RESPECT TO THE INFORMATION IN THIS PUBLICATION, AND SPECIFICALLY DISCLAIMS IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

June 2012

Contents

Preface

About This Guide

This guide describes how to add log collection and analysis support for event sources that the RSA enVision platform does not support. It is intended for administrators and other trusted personnel. Do not make this guide available to the general user population.

RSA enVision Documentation

For information about the RSA enVision platform, see the following documentation:

Release Notes. Provides information about what is new and changed in this release, as well as workarounds for known issues. The latest version of the *Release Notes* is available on RSA SecurCare Online at **https://knowledge.rsasecurity.com**.

Overview Guide. Provides an introduction to RSA enVision platform features and capabilities.

Hardware Setup and Maintenance Guide. Provides instructions on setting up and maintaining RSA enVision appliances. Intended audience is the system administrator.

Configuration Guide. Provides instructions on configuring an RSA enVision site. Intended audience is the system administrator.

Migration Guide. Provides instructions on migrating data from a previous version of the RSA enVision platform to the current version.

Virtual Deployment Guide. Provides instructions on installing an RSA enVision single appliance site or Remote Collector on a virtual infrastructure.

Administrator's Guide. Provides instructions on the basic setup and maintenance of the RSA enVision platform. Includes instructions for the most common administrator tasks.

User's Guide. Provides information that helps users to get started using the RSA enVision platform. Includes instructions for the most common user tasks.

Backup and Recovery Guide. Provides instructions on backing up an RSA enVision system and recovering from a hardware failure.

Security Configuration Guide. Provides an overview of security configuration settings in the RSA enVision platform.

Universal Device Support Guide. Describes how to add log collection and analysis support for event sources that the RSA enVision platform does not support.

RSA enVision Help. Provides comprehensive instructions on setting up RSA enVision processing options and using RSA enVision analysis tools.

RSA continues to assess and improve the documentation. Check RSA SecurCare Online for the latest documentation.

Related Documentation

For information about the RSA enVision Event Explorer module, see the following documentation:

***Release Notes*.** Provides information about what is new and changed in this release, as well as workarounds for known issues.

***Installation Guide*.** Provides instructions on installing the RSA enVision Event Explorer module on your client machine in separate guides for Microsoft Windows and Apple Macintosh operating systems. Intended audience is the end user.

RSA enVision Event Explorer Help. Provides comprehensive instructions on setting up and using the RSA enVision Event Explorer module.

For information about the RSA enVision EventSource Integrator, see the following documentation:

Release Notes. Provides information about what is new and changed in this release, as well as workarounds for known issues.

Overview Guide. Provides an introduction to RSA enVision EventSource Integrator features and capabilities.

RSA enVision EventSource Integrator Help. Provides comprehensive instructions on using RSA enVision Event Source Integrator.

Support and Service

RSA SecurCare Online	**https://knowledge.rsasecurity.com**
Customer Support Information	**www.rsa.com/support**
RSA Secured Partner Solutions Directory	**www.rsasecured.com**

RSA SecurCare Online offers a knowledgebase that contains answers to common questions and solutions to known problems. SecureCare Online also offers information on new releases, important technical news, and software downloads.

The RSA Secured Partner Solutions Directory provides information about third-party hardware and software products that have been certified to work with RSA products. The directory includes Implementation Guides with step-by-step instructions and other information about interoperation of RSA products with these third-party products.

Before You Call Customer Support

Make sure that you have direct access to the computer running the RSA enVision software.

Please have the following information available when you call:

- ❑ One of the following:
  - On a 60-series appliance, the serial number of the appliance.

    You can find the seven-character serial number on the chassis tag on the back of the appliance, or open a Dell Openmanage Server Administrator session, and click **System > Properties > Summary** to find the serial number in the chassis service tag field.
  - On a virtual appliance, the serial number of the RSA enVision software.

    Open the **C:\WINDOWS\system32\drivers\etc\Nie-oe.dat** file, and locate the line that begins with “S/N=”.
- ❑ RSA enVision software version number.
- ❑ The name and version of the operating system under which the problem occurs.
- ❑ On a virtual appliance, the VMware ESX or ESXi server details.

1 Universal Device Support

The RSA enVision platform supports a large number of devices (also referred to as event sources) and is continually adding new devices to the list. However, if you need to collect data and report and alert on events for devices that are not on the list, you can add log collection and analysis support for the device to your system using the Universal Device Support (UDS) feature.

Universal Device Support Tasks

Complete the following tasks to collect and analyze logs from a new device:

1. Plan the methodology of how the RSA enVision platform interprets the syslog messages from the device. For information on planning the device, see "Plan Device Interpretation."
2. Collect the device data. If you cannot collect the logs, you cannot analyze them. Collection can be as simple as configuring the device to send event messages in syslog format to the enVision platform or more complex when the device only supports other collection methods. For information on setting up the device, see the vendor documentation. For more information on collection, see "Data Collection" on page 10.
3. Set up the device in the enVision platform. On an ongoing basis, the NIC Collector Service interprets the incoming event data streams to discover new devices not currently being monitored. You must set up the data collection and analysis options for the discovered devices.

   For information on setting up the device options, see the Help topic "Manage Monitored Devices."
4. Define the XML file for the device. The XML file maps the device message contents to the enVision database tables. The RSA enVision platform uses the XML file for analysis and reporting. You create and define the device in a staging area and move it to the running system when done with the design:

   a. Create the device XML file using the Universal Device Support Console in the staging area. For step-by-step instructions for using the console, see "Creating an XML File."

   b. Add the device to the running system, using the **commit** command. For details, see "Creating an XML File."

   c. Map the message contents to enVision database tables for analysis and reporting using the enVision NIC Device Markup Language (DML). For information on using DML, see "NIC Device Markup Language."

As you define the XML file in step 4, you should have a clear idea of which reports you want to produce from the enVision platform. For complete information on defining and running reports, see the Reports Module section in the Help.

Data Collection

You must be able to collect the data from a device before you can perform any analysis. Devices have various methods to log data, as well as different log transport protocols to enable third-party solutions to access the data.

UDS provides data collection and analysis functionality for devices that support syslog, SNMP, or log file. If you require the RSA enVision platform to collect data from a device that cannot provide log data in any of these methods, contact Customer Support.

Note: A few products provide their log information through a proprietary protocol. To obtain the logs, a third-party solution must connect to the product using that protocol, utilizing function calls published by the vendor. The RSA enVision platform has out-of-the-box connection services for popular products that use proprietary protocols, such as the Check Point LEA API and Cisco IDS POP and XML. This group of products is not covered by the UDS solution.

Syslog

Syslog is the most popular log transport protocol. Syslog Daemons are available on all UNIX-based systems. These include server systems such as Solaris, Linux, AIX, and HP-UX, as well as various network and security appliances.

The overall configuration to collect log messages from a device through syslog is the easiest of all collection protocols because it does not require any configuration on the RSA enVision platform side. All you have to do is configure the device to send the log by syslog to the enVision platform, and enVision starts collecting the logs. Configuration on the device side changes from one product to the next, but most devices require the following parameters:

- IP address for logging host. This location is the enVision system IP address.
- Logging filters. Some devices can filter which messages to send by using severity levels, facilities, and other parameters as filter conditions.

SNMP

SNMP traps are used by various products to send alerts and notifications triggered by specific conditions. The traps are structured based on a Management Information Base (MIB) available with the product.

SNMP traps are different than syslog because, unlike a syslog message that consists of a single line, an SNMP trap is a data structure that contains a number of fields in a given order. Products that support SNMP traps publish their MIB, which describes that data structure. The RSA enVision platform can accept traps from devices that can produce them, but it needs some more information to create an internal syslog event that represents that trap.

Setup

To configure the enVision platform to collect SNMP traps from a device, you must enter the following information:

- Device Name. What the device is referred to as in enVision.
- Vendor ID. A tag identifying the specific device sending the traps. This information is available in the vendor's MIB.
- Device Tag. A tag added to the syslog message to identify that device. This tag enables enVision to discover the device properly.

By default, enVision collects all the information contained in the trap. You can configure the NIC Trapd Service to collect only specific fields and their order within the generated syslog event. For information on the NIC Trapd Service, see the Help.

Example

Here is an example of an SNMP trap collected from an Entercept IDS sensor:

[0] devpc.NETWORK-INTELLIGENCE.COM
[1] 10.10.31.45
[2] enterprises.5.2.0.4.1.3.0 "1001"
[3] enterprises.5.2.0.4.1.4.0 "High"
[4] enterprises.5.2.0.4.1.5.0 "entercept Console"
[5] enterprises.5.2.0.4.1.6.0 "10/22/2003 2:55:49 PM"

The vendor ID in this example is 5.2.0.4.1. The syslog file from this trap is (with the trap service configured to accept traps from an Entercept sensor and the vendor ID provided):

10.10.31.45 %ENTERCEPT
[0]devpc.NETWORK-INTELLIGENCE.COM[1]10.10.31.45[2]enterprises.
5.2.0.4.1.3.0 "1001"[3]enterprises.5.2.0.4.1.4.0 "High"[4]enterprises.5.2.0.4.1.5.0 "entercept Console"[5]enterprises.5.2.0.4.1.6.0 "10/22/2003 2:55:49 PM"

where 10.10.31.45 is the device IP address.

Log File

Some products log data to a local file on the system. The file usually contains multiple log lines structured similarly and a header line that describes this structure. Single or multiple log files may be available, depending on the way the product separates the data.

Log files are text files that contain structured log messages, typically all sharing the same structure. The messages use a delimiter character to separate the various fields. Some devices provide a header line at the top of the file to describe the structure. Most devices keep a single log file, but you may have a device that logs to multiple files, each containing different log messages. For example, Cisco ACS keeps multiple log files for Accounting, Administration, Authentication, and other purposes.

One important issue that you must resolve when working with these types of devices is how to get the log file to the enVision system. The RSA enVision platform needs the log file to process it and forward the events to the NIC Collector Service. The RSA enVision platform runs an FTP server that can accept the file from the logging system. This can be accomplished in one of two ways:

The device has a built-in method to send the file to a host FTP server. Configure the device to send the log files by FTP and provide the enVision system IP address as the FTP server. Most devices also provide configuration for log file rotation or roll-over.

The device does not have a built-in method to send the file to a host FTP server. You can either use a script on the device side to send the file at a specified interval or use the enVision FTP Agent. For information on using the NIC FTP Agent, see the Help.

Setup

Once the log files can be sent to enVision, you must set up the NIC File Reader Service to process the log files. The NIC File Reader Service scans the log files and generates internal syslog events that the NIC Collector can process for each log message.

To configure the enVision platform to collect log files from a device, you must enter the following information:

Device Name. What the device is referred to as in enVision.

Device Tag. The tag added to the syslog message to identify that device. This tag enables enVision to discover the device properly.

By default, enVision collects all the information contained in the log file. For complete information on the NIC File Reader Service, see the Help.

Example

The following is an example of a log file collected from Cisco ACS:

```
Date,Time,User-Name,Group-Name,Caller-Id,Acct-Flags,elapsed_time,service,bytes_in,bytes_out,paks_in,paks
_out,task_id,addr,NAS-Portname,NAS-IP-Address,cmd
03/03/2003,06:58:53,user2,GTNS,1.1.58.14,start,,shell,,,,,203,,telnet146,1.1.191.10,
03/03/2003,07:00:53,user2,GTNS,1.1.58.14,stop,120,shell,,,,,203,,telnet146,1.1.191.10,
03/03/2003,07:01:09,user2,GTNS,1.1.58.14,start,,shell,,,,,7,,telnet146,1.1.191.20,
03/03/2003,07:01:14,user2,GTNS,,NAS Port re-used,5,,,,,,7,,telnet146,1.1.191.20,
```

The first line is the header line, describing the content of the file. Each following line is a single event. The resulting syslog event for the first log entry in the file is:

```
10.10.1.5 %ACS 03/03/2003,07:01:14,user2,GTNS,,NAS Port
re-used,5,,,,,,7,,telnet146,1.1.191.20,
```

where 10.10.1.5 is the device IP address and %ACS is the device tag assigned to the device.

Products Supporting Multiple Log Protocols

Many devices provide log information in more than one format. For example, a Cisco Content Engine appliance sends web traffic logs using file FTP and its system audit logs through syslog. The RSA enVision platform can easily handle these differences by combining both logs in a single device message definition file. You configure the device to send both logs to enVision. In enVision, you configure the NIC File Reader Service to accept logs from that device. This configuration results in syslog data in two different formats from the device. The XML device definition file is configured to recognize both formats and discover both as a Cisco Content Engine.

For complete information on the NIC File Reader Service, see the Help topic “NIC File Reader Service.”

Multiple Products Installed on the Same Physical System

In some cases, a single system runs multiple applications from which you want to collect logs. A popular case is any application running on top of a Microsoft Windows platform. While you may already collect the logs from the OS itself, you may also be interested in the logs generated by an application running on the same box. The RSA enVision platform handles this situation by allowing you to define this device as Multi-Device. This setting tells the enVision platform to look for more than a single device’s logs from that IP address. As enVision matches log events from other devices from that IP address, it adds the appropriate devices to the discovered list automatically.

For complete information on multiple products installed on the same physical system, see the Help topic “Multiple Device IP Address Assignment.”

2 Plan Device Interpretation

Before adding your own custom support for a device to the RSA enVision platform, you must carefully plan the methodology of how enVision interprets the device messages.

To help you in planning your device, use the Device Template on page 29.

Device Interpretation Planning Tasks

There are three tasks involved in planning your device interpretation:

1. Device identification
2. Message definition.
3. Data parsing.

Device Identification

For each device, you must identify the following:

- "Device Class"
- "Device Name"
- "Device Type (dtype)."

Device Class

All supported devices are grouped into device classes and device subclasses.

The device classes represent the general function of the devices. This categorization provides a framework for organizing source devices and the information from source devices. This organization enables enVision to inform you of the area in your network affected by an attack and to provide you with reports that break down network activity, alerts, and incidents by device class.

Each device class is made up of device subclasses. Each device subclass is assigned to only one device class. If the classification is not explicit, the subclass is assigned to the Security device class.

The following table lists the different device classes and subclasses.

Device Class	Device Subclasses
Security	Access Control
	Anti Virus
	DLP
	Firewall
	IDS
	IPS
	Physical Security
	VPN
Network	Configuration Management
	Router
	Switch
	System
	Wireless Devices
Host	Application Servers
	Load Balancing
	Mail Servers
	Mainframe
	Midrange
	Unix
	Web Logs
	Windows Hosts
Storage	Database
	Storage

Device Name

The device name includes the vendor name and product type, for example, Cisco Content Engine.

Multiple Devices with the Same Operating System

Occasionally, several devices from one vendor run the same OS. This type of OS typically combines the functionality and log message sets of all compatible devices into one application. For example, Cisco has the Internet Operating System (IOS), which runs on routers, switches, wireless access points, and so forth.

The device name in this situation is very important, as the name is used for every future discovered device that runs that OS, regardless of its functionality. In this case, RSA recommends a name such as Cisco IOS (Router, Access Point, Switch).

Device Type (dtype)

The RSA enVision platform identifies device types with device type values known as dtype. The RSA enVision platform assigns the dtype for the device when you create the device using UDS.

Use the dtype value when creating new queries or reports. For information on using the dtype to generate reports and queries, see the Help.

Message Definition

Examine every message that a device can send, and specify the message attributes and classifications.

Message Categories

A message category is a group of messages. Message categories are hierarchical, consisting of up to five levels. The highest level of a message category is the NIC category, for example:

- Attacks
- Config

Each alert category is assigned to a NIC category, creating the second level of message category. For example, a message could be assigned to one of the following alert categories:

- Attacks.Access
- Config.Errors

Each event category is assigned to an alert category. You can have up to three levels of event categories, creating a deeper hierarchy. For example, a message could be assigned to one of the following event categories:

- Attacks.Access.Informational
- Attacks.Access.Informational.Eavesdropping
- Attacks.Access.Informational.Host Based
- Attacks.Access.Informational.Host Based.ODBC
- Attacks.Access.Informational.Host Based.SQL
- Config.Errors.Auditing
- Config.Errors.Uploading

Each level of the message category hierarchy is separated by a period (.). For complete information on message categories, see the Help.

Message Definition Review

Here are raw syslog messages from the Cisco PIX and Cisco Content Engine devices. Examples of the key points covered in this section are specified for each message.

Cisco PIX

The following is a raw system message for the Cisco PIX event source (device).

%PIX-1-106022: Deny TCP connection spoof from 10.10.10.1 to 20.20.20.2 on interface E0

Attribute	Value
Class	Security.Firewall
Device Name	Cisco Pix
Device Type	1 (this predefined number already exists within the system)
Alert Category	1
Alert Level	1 (this is derived from the message: %PIX-1)
Event Category	210 (denied event)
NIC Category	10

Cisco Content Engine

The following is a raw system message for the Cisco Content Engine event source (device).

<6> %CCENGINE-6-0006: 1045756274, 30,10.18.50.100, TCP_HIT/200,38191, GET, http://www.cisco.com/, -, DIRECT/-, application/x-javascript

Attribute	Value
Class	Host.Web Logs
Device Name	Cisco Content Engine
Device Type	26 (this predefined number already exists within the system)
Alert Category	1
Alert Level	6 (this is derived from the message: %CCENGINE-6)
Event Category	currently not assigned
NIC Category	12

Data Parsing

The enVision dynamic parser feature utilizes the NIC Device Markup Language (DML) to identify and process device-specific information from system events collected through syslog. DML uses a series of variables known as parsing flags to identify and parse known messages. Each flag represents a parsing and handling instruction for the identified data within the message. These handling instructions give the parser critical information about what type of data it is, whether parsing is necessary, and what fields the parser needs to update in the database table to add this information.

These parsing flags vary by device type as well as the table to which the data is parsed. A group of the most common flags is valid across all devices and tables and includes information such as Device Address, TimeStamp, Message ID, and Message Type. Other flags are more specific and have meaning to only a few device types or tables. Examples of these flags are Group, Profile, and Project, which are only valid parsing instructions for configuration management devices and messages instructed to be parsed to the User Activity table. Here is a simple example of how a parsing flag is identified within the message definition file (**msg.xml**):

<agent>: [ID 366847 auth.notice] '<action>' succeeded for <**username**> on <**terminal**>
Mar 25 15:04:22 **su**: [ID 366847 auth.notice] 'su root' succeeded for **Joey** on **/dev/pts/2**

Not all flags have a set meaning. Some flags are used purely as a placeholder within the messages. For example, in the following message, a flag is used as a wildcard variable to inform the parser to expect something in that position. In this case, it is the <data> tag. This tag is employed to annotate the value HEX in the actual message.

%PIX-2-106012: Deny IP from 20.20.20.2 to 10.10.10.1, IP options **HEX**
Deny IP from <laddr> to <faddr>, IP options **<data>**

Where to Define Parsing

You define parsing instructions in the device message definition file (**msg.xml**) within the content portion of each message. The content field represents the payload data of each message. The payload is the portion of the message that contains the meaning for the system message and includes information such as addresses, ports, reason the event was sent, and connection IDs.

When defining parsing instructions for a message, you should understand the types of data that you can define using the DML parsing flags. Data of interest is classified as data that would be useful within a reporting context, mainly information such as addresses, ports, interfaces, device names, Access Control List names, or Rule names. You can also use DML to define summarization functions using system-defined or user-defined logic. This allows you to define a specific field (key) on which to summarize, for example, summarize connection counts by the **Connection_ID** field.

For a list of the predefined summarization buckets, see “Summaries” on page 63.

Anatomy of a Message Entry in the XML File

This section describes the components of messages.

Example

The following entry is taken from the message definition file for the Cisco PIX Firewall (**ciscopixmsg.xml**).

%PIX-2-106007: Deny inbound tcp from 10.10.10.1/10 to 20.20.20.2/20 due to DNS FLAG
%PIX-2-106007: Deny inbound UDP from 12.15.105.5/8 to 192.168.1.5/9 due to DNS flag.

The following XML MESSAGE element defines how these messages should be processed by the NIC system.

```
<MESSAGE
    category="6"
    level="2"
    parse="1"
    parsedefvalue="1"
    tableid="9"
    id1="106007"
    id2="106007"
    summary="NIC_B_FW_ADDR_ACCOUNTING;fields=0,0;
        sumtype=ciscopix_denyedOutgoingConn;|NIC_B_DEVICES;"
    content="<@inout:*DIRCHK(faddr)>Deny inbound <protocol> from
        <faddr>/<fport> to<laddr>/<lport> due to DNS
        <flag><@ntype:5><@ntype:*securityDirOutBound(inout)>" />
```

Component	Description
Category	Identifies the alert category of the message. In this example, the message is associated with the Denied Connections alert category. This value is assigned based on the XML developer's knowledge of the device.
Level	Identifies the alert level of the message. Alert level is used by the Alert Monitor and Browser. In this example, the message is defined as a critical network event, for which the cause of the event must be corrected. This value is assigned based on the XML developer's knowledge of the device.
Parse and Parsedfvalue	Identifies the parsing status of each message. The Parse tag represents the Current Setting and the Parsedefvalue represents the Original Setting. These tags allow you to add or remove parsing by message ID and revert back when needed. Valid values are 1 (on) and 0 (off). In this example, the value 1 indicates that the message should be parsed.

Component	Description
Tableid	Identifies which table to update when parsing is enabled for a message. These values are located in the **device.ini** file for currently supported devices. This file contains a mapping of table ID number to Actual DB table name. In this example, the value 9 indicates TableList9=FireWall Security and RealTable9=security.
id1	NIC message ID. Matches the value for the vendor message ID (id2), however, if a message can have more than one format, the NIC message ID is used to internally identify different variants of the message. For more information on message IDs, see “Message Identification” on page 22.
id2	Vendor message ID. This ID is always contained somewhere within the syslog message. For more information on message IDs, see “Message Identification” on page 22.
Summary	Identifies the summary information that the message generates.
Content	Contains the expected payload for a particular message ID. Parsing flags are used within this field. Within the XML files, the flag is represented as a value bracketed by the < and > characters. These characters represent the < and > symbols, where < = < and > = >. For more information on message content, see “Parsing and Processing Message Information”on page 23.

In this example, the message is set to parse=true and the Destination table is the Firewall Accounting table (tableid=“6”). The parsing flags associated to this message are in red text. The flags are all associated to information that is valuable within an Accounting report.

<MESSAGE
level="6"
parse="1"
parsedefvalue="1"
tableid="6"
id1="302014"
id2="302014"
sitetrack="1"
content="Teardown TCP connection <accountid> for <finterface>:<faddr>/<fport> to <linterface>:<laddr>/<lport> duration <duration> bytes <bytes> (<username>)" />

By default, enVision assigns categories to every message that can be either explicitly defined for every message that is using the proper tags in the **msg.xml** file or that has been given the default attributes. For example, the default attributes given to every PIX messages are:

- category="0"
- reportcategory="0"
- sitetrack="0"

These values remain at the default setting unless an explicit instruction is given to the message. For example, to make a message available to SiteTrack, the sitetrack="1" tag must be present.

Message Identification

Each message has a NIC message ID (id1) and a vendor message ID (id2). By default, the NIC message ID is the same as the vendor message ID. However, if a message can have more than one format, the NIC message ID is used to internally identify different variants of the message.

The NIC message ID (id1) consists of the vendor message ID and a NIC message ID, separated by a colon, for example, 410001:02.

In the following example, <msgid> 410001 has four possible formats, so the NIC message ID (id1) has the following values: 410001, 410001:01, 410001:02, and 410001:03.

```
<MESSAGE
        category="6"
        level="3"
        parse="1"
        tableid="21"
        id1="410001"
        id2="410001"
        content="UDP DNS packet dropped due to compression length check of <n> bytes:
          actual length:<n1> bytes<@ntype:15><@msg:*RMQ($MSG)>" />
<MESSAGE
        category="6"
        level="3"
        parse="1"
        tableid="21"
        id1="410001:01"
        id2="410001"
        content="UDP DNS packet dropped due to domainname lengthcheck of 255 bytes:
          actual length:<n> bytes<@ntype:15><@msg:*RMQ($MSG)>" />
<MESSAGE
        category="6"
        level="3"
        parse="1"
        tableid="21"
        id1="410001:02"
        id2="410001"
        content="UDP DNS packet dropped due to label length check of 63 bytes actual
          length:<n> bytes<@ntype:15><@msg:*RMQ($MSG)>" />
<MESSAGE
        category="6"
```

```
level="3"
parse="1"
tableid="21"
id1="410001:03"
id2="410001"
content="UDP DNS packet dropped due to packet length check of <n> bytes: actual
  length:<n1> bytes<@ntype:15><@msg:*RMQ($MSG)>" />
```

The order in which these message formats are placed within the XML may be important. If a message format is an exact substring within another message definition, the longest message format definition must come first within the XML.

Parsing and Processing Message Information

The content component of each XML MESSAGE element describes how a message is to be parsed and processed into the enVision platform.

The content component can be broken down into two entities:

- Static text. Static text represents portions of the message that are always present and are the same each time that the message is received. Static data is not saved.
- Parameters. Parameters represent the parts of the message that can be different each time a message is received. Parameters, sometimes known as variables, are defined by an alphanumeric word enclosed within less and greater than symbols, <parameter>. The <parameter> values are deemed to hold all the significant message information. Only <parameter> values can be written into the enVision database.

Parameter Population

A <parameter> can be populated directly from message content, or it can be populated by data derived from the message content.

The following table shows examples of parameter population.

Action	Detail
System receives message	%PIX-2-106007: Deny inbound UDP from 12.15.105.5/8 to 192.168.1.5/9 due to DNS flag.
System processes message based on content description	content="<@inout:*DIRCHK(faddr)>Deny inbound <protocol> from <faddr>/<fport> to<laddr>/<lport> due to DNS <flag><@ntype:5> <@ntype:*securityDirOutBound(inout)>"
System populates parameters	<faddr>, <fport>, <laddr>, <lport>, and <flag> parameter values are populated directly from messages content. Parameters <inout> and <ntype> are populated by values that are derived from the messages content

The system populates parameters in two passes:

1. Populates all parameter values.

   For the previous example, <faddr>, <fport>, <laddr>, <lport>, and <flag> parameter values are populated directly from the message content. <ntype> is set to the predefined value of DeniedInbound(5).

2. Executes all XML conditional statements and utilities in a left-to-right order.

   Because all variables have been populated by the first pass, XML conditional statements and utilities can reference any parameter value in the message.

   For the previous example, the value of <inout> is set, and the value of <ntype> may be modified:

   - <inout> is populated using the DIRCHK utility, <@inout:*DIRCHK(faddr)>.

     Here, the <faddr> value is used as input to the DIRCHK function. DIRCHK determines if <faddr> is located inside or outside of the Enterprise/AS. If <faddr> is located outside of the Enterprise, the target parameter <inout> is set to the Enterprise OUTBOUND direction. Enterprise OUTBOUND is defined as 1, and Enterprise INBOUND is defined as 0.

   - The conditional expression <@ntype:*securityDirOutBound(inout)> is executed. The expression securityDirOutBound is defined as:

     ```
     <REGX
      name="securityDirOutBound"
      parms="parm1"
      default="$NONE"
      expr="<parm1>(1) ? '8'" />
     ```

     If the input parameter's value is OUTBOUND(1), then target parameter <ntype> is set to DeniedOutbound(8).

     Else, the input parameter's value is not OUTBOUND(1), then the REGX default value is used. In this example, the default is defined to the keyword $NONE. $NONE indicates there is no default value, and that the target value should not be modified. Therefore, <ntype>'s value will remain DeniedInbound(5), which was set in pass 1.

Mapping Message Groups to a Table

You can map message groups to tables. Each task in the following procedure is described in more detail in this section.

To map message groups to a database table:

1. After identifying the unique messages, group together messages with similar content.
2. Based on the type of the messages, select the correct table.
3. Define the message and add parsing instructions.

Grouping Messages by Content

You can group messages that have a common theme. For example, all of the following messages deal with Cisco PIX system information:

```
%PIX-1-101002: (PRIORITY) Bad failover cable.
%PIX-1-101005: (PRIORITY) Error reading failover cable status.
%PIX-1-103004: (PRIORITY) Other firewall reports this firewall failed.
%PIX-1-103005: (PRIORITY) Other firewall reporting failure.
%PIX-1-104001: (PRIORITY) Switching to ACTIVE (cause: REASON).
```

You can also group messages together to analyze the content or payload portion of each message created when certain actions are taken by the user, for example, removing a cable, logging on and logging off, or typing a password incorrectly. Notice patterns developing in the order of how messages are sent or that the same messages are sent when particular events occur on the device. This information helps you to identify which messages need to be parsed and which offer redundant information. This also helps you to identify the type of messages that they are, for example, Accounting, Security, Authorization, and so forth.

For some messages that would require parsing if taken alone, when you analyze the pattern in which they occur, you find that the messages produce redundant data. An example of this is the Cisco PIX Build-Up and Teardown messages. The 305011 message represents a successful Build-Up of a connection, however, when you further analyze it, you notice the 305011 message is actually not useful because the reportable data is contained in the 302013 Build-Up message.

The following example illustrates how the 305011 and 305012 messages are redundant messages in terms of accounting information. These messages represent the initial connection between the Local Address and the External facing interface of the PIX. The actual accounting message is the 302014 message, which represents the connection between both endpoint devices and contains a Local and Foreign Address, as well as a connection ID.

Build Up Messages: (Created at Connection Time)

Jul 01 14:22:47 [172.16.8.7] %PIX-6-305011: Built dynamic TCP translation from inside:172.20.21.105/2279 to outside:12.35.22.135/23096
Jul 01 14:22:47 [172.16.8.7] %PIX-6-302013: Built outbound TCP connection 870967 for outside:155.181.136.45/443 (155.181.136.45/443) to inside:172.20.21.105/2279 (12.35.22.135/23096)

Teardown Messages: (Created at Disconnect Time)

Jul 01 14:22:47 [172.16.8.7] %PIX-6-302014: Teardown TCP connection 870967 for outside:155.181.136.45/443 to inside:172.20.21.105/2279 duration 0:00:01 bytes 1606 TCP Reset-I
Jul 01 14:23:15 [172.16.8.7] %PIX-6-305012: Teardown dynamic TCP translation from inside:172.20.21.105/2279 to outside:12.35.22.135/23096 duration 0:00:31

Color	Represents
Green	Translated or Mapped Address of the External interface on the PIX through which the connection went
Blue	Connection ID
Red	Local Address and Port of the Connection

Selecting a Table

Identify the type of message, for example, Accounting, Security, or System, and which table best supports the information that the message contains. Here are a few rules of thumb to use when evaluating a message for category:

Note: The following message examples were taken from a variety of supported devices such as Solaris, Cisco PIX, NetScreen ScreenOS, Top Layer Attack Mitigator, Cisco SIDS, and Cisco VPN Concentrator.

- Information such as Duration or Byte Count is always defined as information that should be retained or parsed into an Accounting type of table. These messages are also the types of messages that most summarizations are run against.

  Oct 09 15:00:26 [10.10.30.96] %PIX-6-302014: Teardown TCP 1000011 for OUTSIDE: 109.876.543.21/80 to INSIDE:123.456.789.10/80 duration 300 bytes 1000 (JerryS)

  Oct 09 15:00:26 [10.10.30.97] Mar 25 15:04:22 ftpd[480]: [ID 742713 daemon.debug] ASCII data connection for test.pl (192.168.1.57,2138**) (856 bytes)**

- Messages dealing with Successful Connections or Connection Attempts are usually assigned to an Authentication table of some sort. These messages are also prime suspects for the summarization functions that enhance the reports ability to display Total Connections and Connection Attempt status in a timely manner.

  Oct 09 15:00:26 [10.10.30.97] %PIX-6-109005: **Authentication succeeded** for user USERNAME from 20.20.20.2/20 to 10.10.10.1/10 on interface INTERFACE

  Oct 09 15:00:26 [10.10.30.97] %PIX-6-109006: **Authentication failed** for user USERNAME from 20.20.20.2/20 to 10.10.10.1/10 on interface INTERFACE

- Messages containing information such as Unsuccessful Connection attempts or Denied and Rejected Connections usually fall into a Security table. These messages usually represent some sort of unauthorized access attempt or attempted use of a privileged command, unauthorized access to a file, or change of file permissions, and so on.

  Oct 09 15:00:26 [10.10.30.97] %PIX-6-106015: Deny tcp (no connection) from 10.10.10.1/10 to 20.20.20.2/20 flags FLAGS on interface INTERFACE

  Oct 09 15:00:26 [10.10.30.97] Oct 09 15:00:26 [10.10.30.97] Mar 25 15:04:22 su: [ID 810491 auth.crit] 'su root' failed for Jack on /dev/pts/2

- Messages dealing with system-specific information, such as environmental information like Fan status, Temperature, Failed Hard Drives, and Power Cycles, usually are parsed into the System tables.

  Oct 09 15:00:26 [10.10.30.97] %PIX-1-103005: (PRIORITY) Other firewall reporting failure.

  Oct 09 15:00:26 [10.10.30.97] %PIX-1-104001: (PRIORITY) Switching to ACTIVE (cause: REASON).

- Messages dealing with VPN information can be sent to any one of the VPN tables that are available within the product. These tables are specifically designed to hold information that is unique to VPN messages, such as Encryption Keys, Tunnel IDs, and additional Address fields.

  Oct 09 15:00:26 [10.10.30.97] 00001 07/21/2003 12:23:29.750 SEV=6 CERT/119 RTP=101 192.168.1.6 The vpndata CRL cache (trusted certificate handle: 64) was cleared successfully

  Oct 09 15:00:26 [10.10.30.97] 00001 07/21/2003 12:10:33.750 SEV=6 AUTH/24 RTP=101 192.168.1.6 Tunnel to headend device 192.168.80.6 connected

- Messages dealing with Intrusion Events have a few tables specifically designed to hold their information. For traditional Intrusion Detection System Events, the IDS table is a good fit, however, for newer multifunction Intrusion Detection and Prevention Systems, it may be easier to fit all of the information contained in those messages into one of the IPS tables.

  Oct 09 15:00:26 [10.10.30.97] Jan 1 2003 08:42:27: %IDSS-2-1205: IP Fragment Too Many Datagrams & from 192.168.1.129 to 209.67.241.202

Oct 09 15:00:26 [10.10.30.97] id=060002 pt=TLN-SEC prot=udp cip=10.20.30.40 cprt=4257 sip=20.30.40.50 sprt=2002 atck="Fragment Restrictions" disp=mitigate ckt=2 src=outside msg="Overlay Header"

- The Level table is a catch-all table for messages that do not contain many individual variables for parsing but have some value to users. The Level table contains a field called msg, which contains the entire payload string of the event. This field enables you to create custom reports that display the contents of this field.

  Oct 09 15:00:26 [10.10.30.97] 00001 07/21/2003 12:24:32.750 SEV=4 CONFIG/17 RTP=101 192.168.1.6 Done writing configuration file ver4.0.1.vpn.image.zip.

  Oct 09 15:00:26 [10.10.30.97] %PIX-2-108001: SMTP made noop: out OUTCHARS in INCHARS data: DATA.

For detailed information on the database tables, see the Help.

Define a Message and Add Parsing Instructions

Static information does not need to be identified with a parsing flag. This information is constant every time a particular message is received and, therefore, can be explicitly defined in the message definition file (**msg.xml**).

In the following messages, the “Teardown TCP” portion is present in both messages, as well as the “for,” “to,” “duration,” and “bytes.” All of these items can be explicitly defined when setting up the parsing for this message ID.

Note: For the purposes of these examples, all < and > are shown as <> bracket symbols.

Oct 09 15:00:26 [10.10.30.96] %PIX-6-302014: Teardown TCP 1000011 for OUTSIDE: 109.876.543.21/80 to INSIDE:123.456.789.10/80 duration 300 bytes 1000 (JerryS)

Oct 09 15:00:29 [10.10.30.96] %PIX-6-302014: Teardown TCP 1230000 for OUTSIDE:123.456.789.11/23 to DMZ:123.456.789.12/5004 duration 600 bytes 75045 (CosmoKramer)

Teardown TCP connection <accountid> for <finterface>:<faddr<fport> to <linterface>:<laddr>/<lport> duration <duration> bytes <bytes> (<username>)

The fields flagged for parsing are:

- accountid = Connection ID of the message
- finterface = Foreign Interface
- faddr = Foreign Address
- fport = Foreign Port
- linterface = Local Interface
- laddr = Local Address
- lport = Local Port
- duration = Duration of the connection
- bytes = Bytes Passed

- username = Username associated to this connection

The Parser understands the concept of “this or that.” For example, the parser interprets all items contained within braces ({}) and separated by a pipe symbol (|) as “this or that” conditions.

In the following example, the Parser expects an end result of added, deleted, or modified, as annotated with the { added | deleted | modified } instruction in the XML file.

FORTRESS: NetScreen device_id=FORTRESS system-notification-00001: ZoneOne address outside with ip address 209.67.241.202 has been added

<data>: NetScreen device_id=<data> { [<vsys>]system | system }-<level>-<msg_id> { (<data>): | : } <src_zone> address <interface> with ip address <saddr> has been { added | deleted | modified }

Device Template

Use the Device Template when planning your device support. You must complete the Device Template prior to creating the XML file.

Device Identification	Value
Device Vendor	
Device Name	
Device Class.Subclass	
Device Type (dtype)	

Messages (list the device messages – grouped by content)

Parsing Instructions (complete this table for each message entry in the xml file)	
Message (use bold text to indicate the fields to be parsed)	
Category	
Database Table	
SiteTrack	

3 Creating an XML File

The XML file maps the device message contents to the RSA enVision platform database tables. The RSA enVision platform uses the XML file for analysis and reporting. Use the Universal Device Support Console to create the XML file for a new user-defined device.

Note: Complete the "Device Template" on page 29 before creating the XML file.

To create the XML file for the device, use the Universal Device Support Console, mapping the message contents to enVision database tables for analysis and reporting. To map the message contents, you must do the following:

1. Understand what the messages are and how they are formatted.

   Each device logs different types of events, formatted in many possible ways. To configure the enVision platform to analyze these messages, you must understand what these messages are and how they are formatted. To understand the message formats, see the vendor documentation.

2. Parse each message to one of the enVision database tables.

   Configure each log message using the enVision NIC Device Markup Language (DML). The message set comprises a descriptor file that instructs the enVision platform on how to interpret all the log messages generated by the device and how to parse the data to the appropriate tables and fields in enVision. For information on using DML, see "NIC Device Markup Language."

3. Summarize useful data.

   Define counts and summaries of multiple events occurring on the network, for example, a total connection count by hour, for appropriate log messages using the NIC Device Markup Language (DML). The RSA enVision platform uses the summarized data to produce accurate and fast reports and queries. For information on using DML, see "NIC Device Markup Language."

This topic includes an example, which assumes the following information:

- device name: foo
- device type: firewall
- sample syslog:

  Oct 09 16:32:17 [192.1.1.1] %CERT-1-101001: (PRIORITY) Failover cable OK.
  Oct 09 16:32:17 [192.1.1.1] %CERT-1-101002: (PRIORITY) Bad failover cable.
  Oct 09 16:32:17 [192.1.1.1] %CERT-1-101003: (PRIORITY) Failover cable not connected (this unit)
  Oct 09 16:32:17 [192.1.1.1] %CERT-1-101004: (PRIORITY) Failover cable not connected (other unit)

Oct 09 16:32:17 [192.1.1.1] %CERT-1-101005: (PRIORITY) Error reading failover cable status.

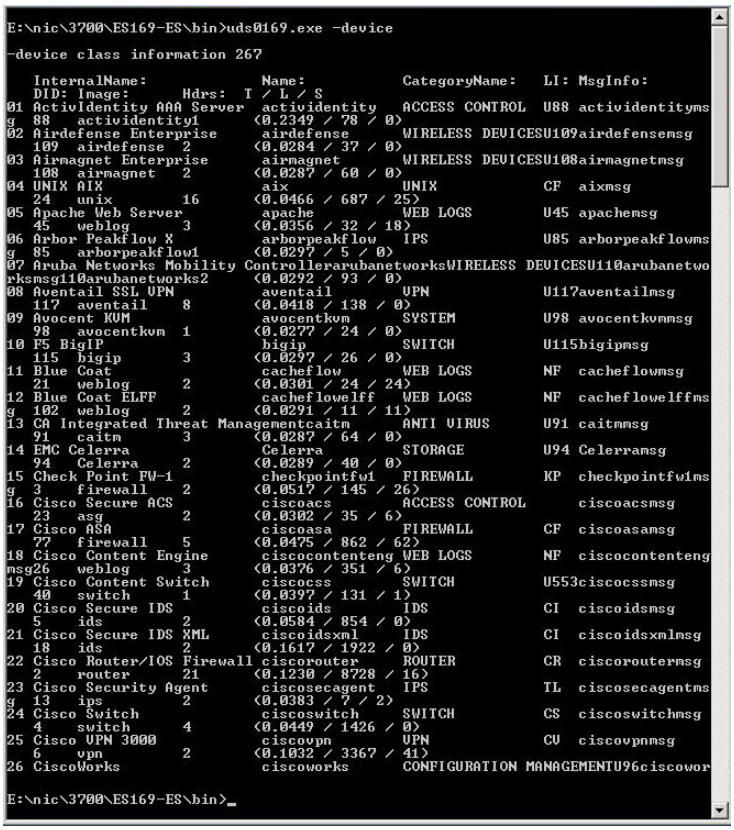
Create XML File

You must create an XML file to process messages for a device.

To create the XML file for the device:

1. Access a DOS Command Prompt. Type **E:\nic\version*nodename*\bin** and press ENTER.
2. Create the device:
   a. Type **uds -device** and press ENTER to verify that the device that you are creating does not already exist.

      The console displays a list of existing devices.



   b. Type the following and press ENTER to add the device and assign it to its associated category:

      uds –create ***devicename*** –category ***deviceclass***

      where:

      - *devicename* is the name of the device to create (valid values are lowercase without spaces).

- *deviceclass* is the name of the category or class associated with the device (case sensitive).

The device name displays on the UI and in reports for this device. This name must be a combination of vendor and product names, such as Cisco Router and Check Point FW1.

The console displays the following messages (the device name in this example is foo):

```
Validating if foo already exists in currently supported device names.
Validating if "Firewall" exists in supported categories
Creation of device foo category Firewall proceeding
Creation of foo succeeded
```

3. The system creates skeleton files in a staging area, **E:\nic\version*nodename*\uds\devices*devicename***, and all work is done out of that staging area.

   After the device definition files are complete, they must be committed, at which point they are placed into the production area, **E:\nic\version*nodename*\etc\devices*devicename***, where they can be used by enVision.

   After you commit the files, you must manually replicate them to the same folder on the other appliances in your site or sites, and you must restart all appliances for the new files to take effect.

   The system creates the following:

   - All directory structures required for the device.
   - All associated files required, in the **E:\nic\version*nodename*\uds\devices*devicename*** directory (where *devicename* is the name of the device you created):
     - ***devicename*.ini**—configuration parameters for the device. You can modify the DisplayName= parameter to a custom value. Do not modify the DatabaseName= parameter.
     - ***devicename*msg.xml**—formatted XML file containing the interpretation code for all of the device messages.
     - ***devicename*client.txt**—text file containing your custom alert description and recommended actions. Edit this file using the enVision UI after adding the new device. For information on editing the alert description and action, see the Manage Messages section in the Help.
     - ***devicename*vendor.txt**—text file containing vendor-supplied alert descriptions and recommended actions.

   The ***devicename*msg.xml** file contains two entries: HEADER and MESSAGE.

   Here is the foomsg.xml file from the example device:

```
<?xml version="1.0" encoding="ISO8859-1" ?>

<DEVICEMESSAGES>

<!--
If the message tag does not contain a definition of a property, the default value will be used.
The default values are:
```

```
        category="0"
        level="1"
        parse="0"
        parsedefvalue="0"
        tableid="1"
        id1=""
        id2=""
        content=""
        reportcategory="0"
        sitetrack="0"
```

The following are the entity reference for all the predefined entities:

```
&lt; < (opening angle bracket)
&gt; > (closing angle bracket)
&amp; & (ampersand)
&quot; " (double quotation mark)
```

```
<HEADER
id1="0001"
id2="0001"
content="&lt;messageid&gt;: &lt;@level:0&gt; &lt;!payload&gt;" />
```

```
<MESSAGE

    category="0"
    level="6"
    parse="1"
    parsedefvalue="1"
    id1="0001"
    id2="0001"
    content="New &lt;message&gt; to &lt;source&gt;" />
```

```
</DEVICEMESSAGES>
```

4. The <HEADER /> section identifies what the message is and where it starts. Edit the Header section for your device:

   a. Use a file editor, such as Notepad, to open the **E:/nic/version/*nodename*/uds/*devicename*/*devicename*msg.xml** file.

   b. Add the header information, according to the methodology that you planned in the Device Template. For details, see "Device Template" on page 29.

      The following is an example of a completed Header section, using the syslog for the example device foo:

```
<HEADER
id1="0001"
id2="0001"
content="%CERT- &lt;level&gt; - &lt;messageid&gt; : &lt;!payload&gt;" />
```

   c. Save the file.

      Note: Windows checks your file for correct XML syntax. Open the .xml file from Explorer and if there are syntax errors, Windows displays a message stating what the errors are.

   d. Type the following and press ENTER to parse the syslog (located in the **logfiles** directory) against the header entries in the XML file to determine if there are errors in these sections. Correct any errors.

uds -parse -header -error

Note: Enclose comments within the XML file between these characters: <!-- -->.

5. The <MESSAGE /> section identifies each message. Edit the Message section for your device:

   a. Use a file editor, such as Notepad, to open the **E:/nic/version/*nodename*/uds/*devicename*/*devicename*msg.xml** file.

   b. Add the information for each message, according to the methodology that you planned in the Device Template. For details, see "Device Template" on page 29.

   Here is an example of a completed Message section, using the syslog for the example device foo:

```
<MESSAGE
    category="0"
    level="6"
    parse="1"
    parsedefvalue="1"
    id1="101001"
    id2="101001"
    content="&lt;(PRIORITY)&gt; Failover cable OK." />
<MESSAGE
    category="0"
    level="6"
    parse="1"
    parsedefvalue="1"
    id1="101002"
    id2="101002"
    content="&lt;(PRIORITY)&gt; Bad failover cable." />
<MESSAGE
    category="0"
    level="6"
    parse="1"
    parsedefvalue="1"
    id1="101003"
    id2="101003"
    content="&lt;(PRIORITY)&gt; Failover cable not connected (this unit)" />
<MESSAGE
    category="0"
    level="6"
    parse="1"
    parsedefvalue="1"
    id1="101005"
    id2="101005"
    content="&lt;(PRIORITY)&gt; Error reading failover cable status." />
<MESSAGE=
```

```
category="0"
level="6"
parse="1"
parsedefvalue="1"
id1="102001"
id2="102001"
content="&lt;(PRIORITY)&gt; Power failure/System reload other side." />
```

c. Save the file.

Note: Windows checks your file for correct XML syntax. Open the .xml file from Explorer and if there are syntax errors, Windows displays a message stating what the errors are.

d. Type the following and press ENTER to parse the syslog (located in the **logfiles** directory) against the message entries in the XML file to determine if there are errors in these sections. Correct any errors.

```
uds -parse -msg -error
```

6. Continue making modifications to the files, and test those changes against actual data.
7. Once you are ready to add the device to the running system, run the **commit** command:

```
uds -commit devicename
```

This command moves the new device files from the staging area to the **E:\nic*version**nodename*\etc\devices*devicename*** directory. After you commit the files, you must manually replicate them to the same folder on the other appliances in your site or sites, and you must restart all appliances for the new files to take effect.

Universal Device Support Console Commands

There are many functions that you can perform using the UDS console. To retrieve this list of commands from within the console itself, type **uds** and press ENTER.

Note: The commands are case sensitive.

General Commands

The following table describes the UDS general-purpose commands.

Command	Description
-log	Outputs data to **envision/deviceiparse.log** file.
-device	Displays all currently supported devices in the system.
-category	Displays all available device categories.

Command	Description
-commit *devicename*	Moves content designed in the **uds\devices** folder for ***devicename*** from the **uds\devices** folder to the **etc\devices** folder.
-device *devicename*	Displays the specified device.
-distr -device *devicename*	Displays variable distribution for the specified device.
-table	Displays all the tables for all devices.
-table *tablename*	Displays the specified device table and its associated variables.
-table *tablename* -tabdist	Displays parameters associated with the specified table.
-table *tablename* -parm	Displays all parameters (both used and not used) associated with the specified table.
-table *tablename* -parm all	Displays all parameters (both used and not used) associated with the specified table.
-table *tablename* -parm used	Displays all the parameters (those that are used) associated with the specified table.
-uvar -device *devicename*	Displays the variables for the specified device.

Create New Device Command

Use the following command to create a new device.

Command	Description
-create *devicename* -category *categoryname*	Creates the device, where *devicename* is the name of the device to create. Valid values are lowercase without spaces. Associates the *devicename* with the *categoryname*, where *categoryname* is the name of the category associated with the device (case sensitive).

Parse Data of a Defined Device Type Commands

The following table describes the commands that extract information from the device XML files.

Command	Description
-parse	Parses out time, IP address, level, and message.
-parse db	Parses data out of the message based on database table parameter.
-parse error	Display output of any message found that cannot be parsed.

NIC Server Data Retrieval Commands

The following table describes the commands that retrieve data from the NIC Server.

Command	Description
-nsdata IPaddress	Retrieves log data from the NIC Server in real time. The console waits for data to arrive and processes it. For example: uds -device ciscopix -error -table ACCOUNTING -parse -msg -nsdata 110.110.0.1
-nsdata IPaddress:StartTime:EndTime	Retrieves log data from the NIC Server for a specific time range. For example: uds -device ciscopix -error -table ACCOUNTING -parse -msg –nsdata 110.110.0.1:20031204110121: 20031204120121

Examples

This section lists examples for some of the preceding commands.

Example 1

This example displays information for the **zorp** device.

Jul 28 13:15:46 [11.22.333.444] auli zorp/intra[2443]: core.error(1): Error receiving datagram on listening stream; fd='2', error='bug'

<MESSAGE level="6" parse="1" parsedefvalue="1" tableid="37" id1="core_error_1" id2="core_error_1" eventcategory="1605010000" content="Error receiving datagram on <interface>; fd='<protocol>', error='<reason>' <@action:Error receiving datagram><@fhost:*HDR(hhost)><@username:*HDR(hinstance)><@client_port:*HDR(hsessionId)>" />

uds.exe -device zorp -parse -msg -verbose

-device class information

InternalName:	Name:	CategoryName:	LI:	MsgInfo:	DID:	Image:	Hdrs:	T / L / S
01 UNIX Solaris	zorp	UNIX	CF	zorpmsg	619	unix	1	(0.8459 / 3 / 0)

-parse:

Date/Time,Device,Level,MsgID,action,client_port,duration,fhost,interface,ntype,protocol,rbytes,reason,sbytes,username
Unknown IP / Device XML mapping, attempt to discover device
Jul 28 13:15:46 [11.22.333.444] auli zorp/intra[2443]: core.error(1): Error receiving datagram on listening stream; fd='2', error='bug'

Header: processed by HEADER XML Element (id1) : H01
<hhost> parm :(0-4) : "auli"
<hinstance> parm :(5-15) : "zorp/intra"
[data :(15-16)
<hpid> parm :(16-20) : "2443"
]: data :(20-22)
<msgIdPart1> parm :(23-27) : "core"
. data :(27-28)
<msgIdPart2> parm :(28-33) : "error"
(data :(33-34)
<msgIdPart3> parm :(34-35) : "1"
) data :(35-36)
: data :(36-37)
<!payload> mrkr :(38-103) : "Error receiving datagram on listening stream; fd='2', error='bug'"

Jul 28 13:15:46 [11.22.333.444] auli zorp/intra[2443]: core.error(1): Error receiving datagram on listening stream; fd='2', error='bug'
Header: processed by HEADER XML Element (id1) : H01
<hhost> parm :(0-4) : "auli"
<hinstance> parm :(5-15) : "zorp/intra"
[data :(15-16)
<hpid> parm :(16-20) : "2443"
]: data :(20-22)
<msgIdPart1> parm :(23-27) : "core"
. data :(27-28)
<msgIdPart2> parm :(28-33) : "error"
(data :(33-34)
<msgIdPart3> parm :(34-35) : "1"
) data :(35-36)
: data :(36-37)
<!payload> mrkr :(38-103) : "Error receiving datagram on listening stream; fd='2', error='bug'"
Payload: processed by MESSAGE XML Element (id1) : core_error_1
Error receiving datagram on data :(0-27) Error receiving datagram on
<interface> parm :(28-44) listening stream
; fd=' data :(44-50) ; fd='
<protocol> parm :(50-51) 2
', error=' data :(51-61) ', error='
<reason> parm :(61-64) bug
' data :(64-65) '
<fhost:*HDR:*HDR(hhost)> : auli
<username:*HDR:*HDR(hinstance)> : zorp/intra
<client_port:*HDR:*HDR(hsessionId)> :

Jul 28 13:15:46,[11.22.333.444],0,core_error_1,Error receiving datagram,,,auli,listening stream,,2,,bug,,zorp/intra

Example 2

This example displays information for NIC events of the **Windows** device.

Jan 25 08:52:43 [22.33.44.555] %NICWIN-4-Security_529_Security: Security,10128943,Wed Jan 25 08:50:55 2006 ,529,Security,NT AUTHORITY/SYSTEM,Failure Audit,TESTSERVER,Logon/Logoff ,,Logon Failure: Reason: Unknown user name or bad password User Name: minnie Domain: NTMBC Logon Type: 3 Logon Process: NtLmSsp Authentication Package: NTLM Workstation Name: WE1-MINNIE

<MESSAGE level="4" parse="1" parsedefvalue="1" tableid="5" id1="Security_529_Security" id2="Security_529_Security" eventcategory="1401030000" summary="NIC_B_WINDOWS;sumtype=11;|NIC_B_WINDOWS;key=event_computer;sumtype=12;|NIC_B_WINDOWS;key=event_type;sumtype=13;|NIC_B_WINDOWS;key=category;sumtype=14;|NIC_B_CATEGORIES;sumtype=denied_in;|NIC_B_CATEGORIES;subkey=event_log;sumtype=connection;|NIC_B_CATEGORIES;subkey=username;sumtype=5;" content="<@utcstamp:*UTC($MSG,'%B %D %N:%U:%O %W',datetime)><@category:Logon_Event><@reason:Unknown user name or bad password ><@event_user:*RMQ(event_user)><event_log>,<linenum>,<day> <datetime>,<event_id>,<event_source>,<event_user>,<event_type>,<event_computer>,<category>,<data>,<event_description>: <space> Reason: <space> Unknown user name or bad password <space> User Name: <username> Domain: <domain> Logon Type: <logon_type> Logon Process: <process> Authentication Package: <auth_package> Workstation Name: <workstation>" />

uds.exe -device winevent_nic -parse -msg -verbose

-device class information

InternalName: Name: CategoryName: LI: MsgInfo: DID: Image: Hdrs: T / L / S

01 Windows Events (NIC) winevent_nic WINDOWS HOSTS CI winevent_nicmsg 30 winhost 3 (2.2536 / 6627 / 6558)

-parse:

Date/Time,Device,Level,MsgID,accesses,addr,agent,application,auth_package,auth_type,bytes,c_domain,c_logon_id,c_user_name,category,change,client_ip,data,datetime,day,doc_number,document_name,domain,domain_id,event_computer,event_description,event_id,event_log,event_source,event_type,event_user,filter,fld1,fld2,fld3,fld4,fld5,fld6,fld8,fld9,handle_id,inbound_spi,linenum,logon_id,logon_type,misc,misc_id,misc_name,mode,num_pages,number,obj_handle,obj_server,obj_type,operation_id,options,outbound_spi,parameters,peer_id,portname,printer,privileges,process,process_id,reason,space,tbdint1,tbdint2,tbdstr1,tbdstr2,tbdstr3,type,user_id,username,utcstamp,

workstation

Unknown IP / Device XML mapping, attempt to discover device

Jan 25 08:52:43 [22.33.44.555] %NICWIN-4-Security_529_Security: Security,10128943,Wed Jan 25 08:50:55 2006 ,529,Security,NT AUTHORITY/SYSTEM,Failure Audit,TESTSERVER,Logon/Logoff ,,Logon Failure: Reason: Unknown user name or bad password User Name: minnie Domain: NTMBC Logon Type: 3 Logon Process: NtLmSsp Authentication Package: NTLM Workstation Name: WE1-MINNIE

Header: processed by HEADER XML Element (id1) : H01
%NICWIN- data :(0-8)
<hlevel> parm :(8-9) : "4"
- data :(9-10)
<messageid> parm :(10-31) : "Security_529_Security"
: data :(31-32)
<!payload> mrkr :(33-349) : "Security,10128943,Wed Jan 25 08:50:55 2006 ,529,Security,NT AUTHORITY/SYSTEM,Failure Audit,TESTSERVER,Logon/Logoff ,,Logon Failure: Reason: Unknown user name or bad password
User Name: minnie Domain: NTMBC Logon Type: 3 Logon Process: NtLmSsp Authentication Package: NTLM Workstation Name: WE1-MINNIE

Jan 25 08:52:43 [22.33.44.555] %NICWIN-4-Security_529_Security: Security,10128943,Wed Jan 25 08:50:55 2006 ,529,Security,NT AUTHORITY/SYSTEM,Failure Audit,FRANK-HA,Logon/Logoff ,,Logon Failure: Reason: Unknown user name or bad password User Name: minnie Domain: NTMBC Logon Type: 3 Logon Process: NtLmSsp
Authentication Package: NTLM Workstation Name: WE1-MINNIE

Header: processed by HEADER XML Element (id1) : H01
%NICWIN- data :(0-8)
<hlevel> parm :(8-9) : "4"
- data :(9-10)
<messageid> parm :(10-31) : "Security_529_Security"
: data :(31-32)
<!payload> mrkr :(33-349) : "Security,10128943,Wed Jan 25 08:50:55 2006 ,529,Security,NT AUTHORITY/SYSTEM,Failure Audit,TESTSERVER,Logon/Logoff ,,Logon Failure: Reason: Unknown user name or bad password
User Name: minnie Domain: NTMBC Logon Type: 3 Logon Process: NtLmSsp Authentication Package: NTLM Workstation Name: WE1-MINNIE
Payload: processed by MESSAGE XML Element (id1) : Security_529_Security

<event_log> parm :(0-8) Security
, data :(8-9) ,
<linenum> parm :(9-17) 10128943
, data :(17-18) ,
<day> parm :(18-21) Wed
<datetime> parm :(22-43) Jan 25 08:50:55 2006
, data :(43-44) ,
<event_id> parm :(44-47) 529
, data :(47-48) ,
<event_source> parm :(48-56) Security
, data :(56-57) ,
<event_user> parm :(57-76) NT AUTHORITY/SYSTEM
, data :(76-77) ,
<event_type> parm :(77-90) Failure Audit
, data :(90-91) ,
<event_computer> parm :(91-101) TESTSERVER
, data :(101-102) ,
<category> parm :(102-115) Logon/Logoff
, data :(115-116) ,
<data> parm :(116-116)
, data :(116-117) ,
<event_description> parm :(117-130) Logon Failure
: data :(130-131) :
<space> parm :(134-134)
Reason: data :(134-141) Reason:
<space> parm :(143-143)
Unknown user name or bad password data :(143-176) Unknown user name or bad password
<space> parm :(179-179)
User Name: data :(179-189) User Name:
<username> parm :(190-198) minnie
Domain: data :(198-205) Domain:
<domain> parm :(207-215) NTMBC
Logon Type: data :(215-226) Logon Type:
<logon_type> parm :(227-231) 3
Logon Process: data :(231-245) Logon Process:
<process> parm :(246-257) NtLmSsp
Authentication Package: data :(257-280) Authentication Package:
<auth_package> parm :(281-288) NTLM

Workstation Name: data :(288-305) Workstation Name:
<workstation> parm :(306-316) WE1-MINNIE
<utcstamp:*UTC:*UTC($MSG,'%B %D %N:%U:%O %W',datetime)> : 1138197055
<event_user:*RMQ:*RMQ(event_user)> : NT AUTHORITY/SYSTEM

Jan 25 08:52:43,[22.33.44.555],0,Security_529_Security,,,,,NTLM,,,,,,Logon/Logoff,,,,Jan 25 08:50:55 2006,Wed,,,NTMBC,,TESTSERVER,Logon Failure,529,Security,Security,Failure Audit,NT AUTHORITY/SYSTEM,,,,,,,,,,,10128943,,3,,,,,,,,,,,,,,,,,NtLmSsp,,Unknown user name or bad password ,,,,,,,,minnie,1138197055,WE1-MINNIE

Example 3

This example displays information for the **Linux** device.

Nov 09 13:54:26 [12.34.56.789] sshd[32188]: Accepted password for jdoe from 11.22.333.44 port 2786 ssh2

<MESSAGE level="3" parse="1" parsedefvalue="1" tableid="37" id1="00020" id2="sshd" eventcategory="1401030000" summary="NIC_B_CATEGORIES;sumtype=5;" content="<agent>[<data>]: Illegal user <username> from <faddr> <@:*SYSVAL($MSGID,$ID1)><@action:illegal user><@ntype:22>" />

<MESSAGE level="3" parse="1" parsedefvalue="1" tableid="37" id1="00020:01" id2="sshd" eventcategory="1301020000" content="<agent>[<data>]: fatal: Timeout before authentication for <faddr> <@reason:timeout before authentication><@action:authentication failure><@:*SYSVAL($MSGID,$ID1)>" />

<MESSAGE level="3" parse="1" parsedefvalue="1" tableid="37" id1="00020:02" id2="sshd" eventcategory="1301020000" content="<agent>[<data>]: fatal: PAM pam_chauthtok failed[<data>]: Authentication token manipulation error <@reason:Authentication token error><@action:pam_chauthok failure><@:*SYSVAL($MSGID,$ID1)>" />

<MESSAGE level="3" parse="1" parsedefvalue="1" tableid="37" id1="00020:03" id2="sshd" eventcategory="1401030000" summary="NIC_B_CATEGORIES;sumtype=5;" content="<agent>[<data>]: Failed password { for illegal user | for } <username> from <faddr> port <fport> <method> <@ntype:22><@action:failed password><@:*SYSVAL($MSGID,$ID1)>" />

<MESSAGE level="3" parse="1" parsedefvalue="1" tableid="37" id1="00020:04" id2="sshd" eventcategory="1401030000" content="<agent>[<data>]: check pass; user unknown <@reason:user unknown><@action:check pass><@:*SYSVAL($MSGID,$ID1)><@ntype:22>" />

<MESSAGE level="3" parse="1" parsedefvalue="1" tableid="37" id1="00020:05" id2="sshd" eventcategory="1301020000" summary="NIC_B_CATEGORIES;sumtype=5;" content="<agent>[<data>]: authentication failure; logname=<accountid> uid=<id> euid=<id> tty=<interface> ruser=<username> rhost=<faddr> <@action:authentication failure><@:*SYSVAL($MSGID,$ID1)><@ntype:22>" />

<MESSAGE level="6" parse="1" parsedefvalue="1" tableid="37" id1="00020:06" id2="sshd" eventcategory="1301020000" content="<agent>[<data>]: <fport> more authentication { failures; | failure; } logname=<account> uid=<id> euid=<id> tty=<interface> ruser=<tbdstr1> rhost=<fhost> user=<username> <@action:authentication failure><@:*SYSVAL($MSGID,$ID1)>" />

<MESSAGE level="6" parse="1" parsedefvalue="1" tableid="37" id1="00020:07" id2="sshd" eventcategory="1301020000" content="<agent>[<data>]: <fport> more authentication { failures; | failure; } logname=<account> uid=<id> euid=<id> tty=<interface> ruser=<username> rhost=<fhost> <@action:authentication failure><@:*SYSVAL($MSGID,$ID1)>" />

<MESSAGE level="6" parse="1" parsedefvalue="1" tableid="13" id1="00020:08" id2="sshd" eventcategory="1801020000" content="<agent>[<data>]: session opened for user <username> by (uid=<data>) <@ntype:21><@:*SYSVAL($MSGID,$ID1)>" />

<MESSAGE level="6" parse="1" parsedefvalue="1" tableid="13" id1="00020:09" id2="sshd" eventcategory="1801020000" content="<agent>[<data>]: session opened for user <username> <@:*SYSVAL($MSGID,$ID1)><@ntype:21>" />

<MESSAGE level="6" parse="1" parsedefvalue="1" tableid="13" id1="00020:13" id2="sshd"

eventcategory="1801020000" content="<agent>[<data>]: session closed for user <username> <@:*SYSVAL($MSGID,$ID1)>" />

<MESSAGE level="6" parse="1" parsedefvalue="1" tableid="13" id1="00020:10" id2="sshd" eventcategory="1605010000" summary="NIC_B_ADDRESS_ACCOUNTING;sumtype=1;" content="<agent>[<data>]: Accepted { publickey | password } for <username> from <faddr> port <fport> <data><@:*SYSVAL($MSGID,$ID1)><@ntype:21>" />

<MESSAGE level="6" parse="1" parsedefvalue="1" tableid="13" id1="00020:11" id2="sshd" eventcategory="1301020000" content="<agent>[<data>]: log: Closing connection to <faddr> <@:*SYSVAL($MSGID,$ID1)>" />

<MESSAGE level="6" parse="1" parsedefvalue="1" tableid="34" id1="00020:12" id2="sshd" eventcategory="1605010000" content="<message> <@:*SYSVAL($MSGID,$ID1)> <@msg:*PARMVAL($MSG)><@level:*SYSVAL($LEVEL)>" />

c:\envision\bin\uds.exe -device rhlinux -parse -msg -verbose

-device class information

InternalName:	Name:	CategoryName:	LI:	MsgInfo:	DID:	Image:	Hdrs: T / L / S
01 Linux	rhlinux	UNIX	CF	rhlinuxmsg	27	unix	11 (0.8832 / 442 / 38)

-parse:

Date/Time,Device,Level,MsgID,account,accountid,action,agent,data,faddr,fhost,fld,fld1,fld2,fld3,fport,group,id,id2,interface,level,login,logname,message,method,msg,name,ntype,num,phost,protocol,reason,space,tbdstr1,terminal,username

Unknown IP / Device XML mapping, attempt to discover device

Nov 09 13:54:26 [12.34.56.789] sshd[32188]: Accepted password for jdoe from 11.22.333.44 port 2786 ssh2

Header: processed by HEADER XML Element (id1) : H01
<messageid> parm :(0-12) : "sshd[32188]:"
(pam_unix)[data :(0-0)
Header: processed by HEADER XML Element (id1) : H02
[data :(0-0)
Header: processed by HEADER XML Element (id1) : H03
<messageid> parm :(0-4) : "sshd"
[data :(4-5)
<data> parm :(5-10) : "32188"
]: data :(10-12)
<!payload:messageid> mrkr :(13-76) : sshd[32188]: Accepted password for jdoe from 11.22.333.44 port 2786 ssh2
Nov 09 13:54:26 [12.34.56.789] sshd[32188]: Accepted password for jdoe from 11.22.333.44 port 2786 ssh2

Header: processed by HEADER XML Element (id1) : H01
<messageid> parm :(0-12) : "sshd[32188]:"
(pam_unix)[data :(0-0)
Header: processed by HEADER XML Element (id1) : H02
[data :(0-0)
Header: processed by HEADER XML Element (id1) : H03
<messageid> parm :(0-4) : "sshd"
[data :(4-5)
<data> parm :(5-10) : "32188"
]: data :(10-12)
<!payload:messageid> mrkr :(13-76) : sshd[32188]: Accepted password for jdoe from 11.22.333.44 port 2786 ssh2
Error: <!payload> does not match XML Message content string (id1):00020 (id2):sshd
<agent> parm :(0-0)

[data :(4-5)
<data> parm :(0-0)
]: Illegal user data :(0-0)

Remaining <!payload> not processed:

"32188]: Accepted password for jdoe from 11.22.3..."

Error: <!payload> does not match XML Message content string (id1):00020:01 (id2):sshd
<agent> parm :(0-0)
[data :(4-5)
<data> parm :(0-0)
]: fatal: Timeout before authentication for data :(0-0)

Remaining <!payload> not processed:

"32188]: Accepted password for jdoe from 11.22.3..."

Error: <!payload> does not match XML Message content string (id1):00020:02 (id2):sshd

<agent> parm :(0-0)
[data :(4-5)
<data> parm :(0-0)
]: fatal: PAM pam_chauthtok failed[data :(0-0)

Remaining <!payload> not processed:

"32188]: Accepted password for jdoe from 11.22.3..."

Error: <!payload> does not match XML Message content string (id1):00020:03 (id2):sshd
<agent> parm :(0-0)
[data :(4-5)
<data> parm :(0-0)
]: Failed password data :(0-0)

Remaining <!payload> not processed:

"32188]: Accepted password for jdoe from 11.222.3..."

Error: <!payload> does not match XML Message content string (id1):00020:04 (id2):sshd

<agent> parm :(0-0)
[data :(4-5)
<data> parm :(0-0)
]: check pass; user unknown data :(0-0)

Remaining <!payload> not processed:

"32188]: Accepted password for jdoe from 11.222.3..."

Error: <!payload> does not match XML Message content string (id1):00020:05 (id2):sshd

<agent> parm :(0-0)
[data :(4-5)
<data> parm :(0-0)
]: authentication failure; logname= data :(0-0)

Remaining <!payload> not processed:

"32188]: Accepted password for jdoe from 11.22.3..."

Error: <!payload> does not match XML Message content string (id1):00020:06 (id2):sshd

<agent> parm :(0-0)
[data :(4-5)
<data> parm :(0-0)

]: data :(10-12)
<fport> parm :(0-0)
more authentication data :(0-0)

Remaining <!payload> not processed:

"Accepted password for jdoe from 11.22.333.44 po..."

Error: <!payload> does not match XML Message content string (id1):00020:07 (id2):sshd

<agent> parm :(0-0)
[data :(4-5)
<data> parm :(0-0)
]: data :(10-12)
<fport> parm :(0-0)
more authentication data :(0-0)

Remaining <!payload> not processed:

"Accepted password for jdoe from 11.22.333.44 po..."

Error: <!payload> does not match XML Message content string (id1):00020:08 (id2):sshd

<agent> parm :(0-0)
[data :(4-5)
<data> parm :(0-0)
]: session opened for user data :(0-0)

Remaining <!payload> not processed:

"32188]: Accepted password for jdoe from 11.22.3..."

Error: <!payload> does not match XML Message content string (id1):00020:09 (id2):sshd

<agent> parm :(0-0)
[data :(4-5)
<data> parm :(0-0)
]: session opened for user data :(0-0)

Remaining <!payload> not processed:

"32188]: Accepted password for jdoe from 11.22.3..."

Error: <!payload> does not match XML Message content string (id1):00020:13 (id2):sshd

<agent> parm :(0-0)
[data :(4-5)
<data> parm :(0-0)
]: session closed for user data :(0-0)

Remaining <!payload> not processed:

"32188]: Accepted password for jdoe from 11.22.3..."

Payload: processed by MESSAGE XML Element (id1) : 00020:10
<agent> parm :(0-4) sshd
[data :(4-5) [
<data> parm :(5-10) 32188
]: Accepted data :(10-21)]: Accepted
password data :(22-30) password
for data :(31-34) for
<username> parm :(35-43) jdoe
from data :(43-47) from
<faddr> parm :(48-62) 11.22.333.44
port data :(62-66) port

<fport> parm :(67-71) 2786
<data> parm :(72-76) ssh2
<:*SYSVAL:*SYSVAL($MSGID,$ID1)> : 00020:10

Nov 09 13:54:26,[12.34.56.789],0,00020:10,,,,sshd,ssh2,11.22.333.44,,,,,,2786,,,,,,,,,,,,21,,,,,,,,jdoe

4 NIC Device Markup Language

The RSA enVision platform NIC Device Markup Language (DML) is used to create an XML file, which describes data contained in a specific device syslog event.

A syslog event is made up of components that are put together by both the source and destination systems in order to provide notification of an event that occurred on the source system. DML is used to provide a description for the entire message format of a given device's set of possible messages based on specific rules.

The operating copy of the device DML description file is located in the associated production area **E:\nic*version**nodename*\etc\devices*devicename*** directory. The file name is ***devicename*msg.xml**, where *devicename* is the name of the device. For example, a Cisco PIX device support file would be located in the **E:\nic*version**nodename*\devices\ciscopix** directory and be labeled **ciscopixmsg.xml**.

The UDS **-create** command creates (among other files) the device XML file in the staging area, **E:\nic*version**nodename*\uds\devices*devicename*** directory. The UDS **-commit** command moves the completed device files (including the XML) to the production area, **E:\nic*version**nodename*\etc\devices*devicename***.

XML Basics

The following are some basic XML concepts to help you work with the XML file:

- XML code is bound in tags by < and />
  - < starts a tag
  - /> ends a tag
- Replace certain characters between tags with control codes, as shown in this table.

Character to Replace	Character to Use
<	&lt;

Character to Replace	Character to Use
>	>
“	"
‘	$apos;
&	&

- Use comments to explain what the XML code is attempting to do. Enclose comments in special tags:
  - Start comments with **<!--**
  - End tags with -->

  For example, <!-- This is a comment. -->

Device XML

The following is the XML file template for a new device:

```
<?xml version="1.0" encoding="ISO8859-1" ?>

<DEVICEMESSAGES>

<!--
If the message tag does not contain a definition of a property,
the default value will be used.
The default values are:
      category="0"
      level="1"
      parse="0"
      parsedefvalue="0"
      tableid="1"
      id1=""
      id2=""
      content=""
      reportcategory="0"
       sitetrack="0"

The following are the entity reference for all the predefined entities:
    &lt;< (opening angle bracket)
    &gt;> (closing angle bracket)
    &amp;& (ampersand)
    &quot;" (double quotation mark)

 -->
<HEADER
id1="0001"
id2="0001"
content="&lt;messageid&gt;: &lt;@level:0&gt; &lt;!payload&gt;" />
<MESSAGE
      category="0"
      level="6"
```

```
    parse="1"
    parsedefvalue="1"
    id1="0001"
    id2="0001"
    content="New &lt;message&gt; to &lt;source&gt;" />
</DEVICEMESSAGES>
```

Syslog Message Format

A syslog message is made up of three fixed components:

- Date and time stamp
- Source IP address
- Payload

Note: The **syslog.unx** file is located in the **enVison\logfiles** folder.

The payload varies depending on the source device and the format of data that it sends.

The message payload is made up of two elements:

- <HEADER />
- <MESSAGE />

Each message element is a description of a message or set of messages that the enVision platform receives from a device. It is made up of a set of value-paired fields, *x=y*, in no specific order. If a value-paired field is not explicitly denoted within the message element, the system uses its default value.

Format conventions for the payload are:

- A single word of text is delimited by spaces or a defined delimiter.
- A parameter can be optionally enclosed in square brackets ([]).
- A parameter enclosed in braces ({}). Braces cannot appear within braces.
- Type of element variable enclosed in angle brackets (<>). This denotes the type, not the exact working of the element in the message. This type maps to a given variable in the associated data class. If a variable is present but not mapped to the associated data class, that variable is ignored.
- Multiple options for an element enclosed in square brackets ([]), braces ({}), or both are separated by a pipe (|). Braces cannot be enclosed within braces.
- A marker is denoted as <! word>. A marker identifies that a position is to be returned and denotes the start of another defined element.
- A force variable word to value is denoted as <@word:value>.

You can specify where a payload begins by using <!payload:var>.

In the following example, the payload begins at the <data> variable (not after the <messageid>). This structure allows UDS to support more complex message formats. The example below shows the <messageid> near the end of the log message, and the !payload near the beginning of the message.

```
<HEADER
 id1="0001"
 id2="0001"
 content=""<data>” <level> <messageid> <!payload:data>;" />
<MESSAGE
  category="0"
  level="6"
  parse="1"
  tableid="15"
  parsedefvalue="1"
  id1="140000:96"
  id2="140000:96"
  content="<msg>” <ntype> <status> ../<h_code>" />
```

The following UDS console output shows <!payload:data> setting the payload to begin at the variable <data> location in the log message.

```
Oct 29 08:56:40 [10.10.50.199]  "ICP: Service disabled."  0 140000:96   ../icp.cpp:292

---------------------------------------------------------------------

Header:
  0:data - """                          --> match
  0:parm - <data> value: "ICP: Service disabled."  --> match
  0:data - """                          --> match
  0:parm - <level> value: "0"                --> match
  0:parm - <messageid> value: "140000:96"         --> match
  0:mrkr - <!payload:data> value: ICP: Service disabled."  0 140000:96   ../icp.cpp:292--> match

Payload: processed by XML Element (id1)      : 140000:96

  <msg>               : t(1) (0-22) = ICP: Service disabled.
  <ntype>             : t(1) (25-26) = 0
  <status>            : t(1) (27-39) = 140000:96
  <h_code>             : t(1) (42-53) = icp.cpp:292
---------------------------------------------------------------------
```

Use the $START tag to reset to the beginning of the message when the message ID is not at the beginning of the message but somewhere in the payload. The parser must reset to the beginning of the message after finding the message ID so that all of the data before the message ID is parsed.

```
<!payload:$START>
```

Header

The header, <HEADER />, is located after the syslog fixed components and before the message identifier. The header identifies what the message is and where it starts.

The following are the two formats for a header:

- < HEADER id = 001 format1 = “<date> <time> <seqnumber>: %<facility>-<severity>-<msgid> <!message>” />

- < HEADER id = 002 format2 = %<facility>-<severity>-<msgid> <!message>” />

The following is an example of the initial line of syslog:

May 07 14:00:00 [192.168.1.202] %PIX-6-305012: Teardown dynamic UDP translation from inside:192.168.1.24/14848 to dmz:192.168.5.254/6159 duration 0:00:31

Variable	Data
SyslogDateTime	May 07 14:00:00
SourceDevice	[192.168.1.202]
facility	PIX
severity	6
msgid	305012
message	Teardown dynamic UDP translation from inside:192.168.1.24/14848 to dmz:192.168.5.254/6159 duration 0:00:31

Header

A header is made up of fixed and optional variables. All variables are delimited from one another, usually by spaces.

The system assumes that, if optional values are present in one message, the values are present in all messages from that device.

Optionally, fixed characters are appended to the end or beginning of a variable. You can configure whether these characters are used as fixed variables within the definition of the message or not included in the variable. For example, Cisco appends a percent character (%) to the front of all facility strings and a colon (:) after the sequence number. Because they are always a part of the message, there is no reason to process these characters.

Fixed Variables

Here are the fixed variables that must be contained within the header element defined in the XML file. In cases where this structure might vary, a specific header is defined within the device XML file.

Fixed Variable	Description
id	Unique identification of this header descriptor for this device type.
msgid	Identification of a specific message and associated variables within the accompanying message body.
message	Start of information associated with the message, up to new line character marking end of line.

Optional Variables

The following are the optional variables that can appear in the header.

Optional Variable	Description
devts	Indicates that a device time stamp exists in the header and identifies the format of the device time stamp.
date	Numeric representation of month, day, and year when the message was sent by the facility.
time	Numeric representation of the hour, minute, and second when the message was sent by the facility.
seqnumber	Sequence number of this event.
facility	Facility to which the message refers. A facility can be a hardware device, protocol, or module of system software. Valid values are two or more uppercase letters.
severity	Message severity. Valid values are 0 – 9.
msgid	Identifies a specific message and associated variables within the accompanying message body.
message	Start of information associated with a message.

Device Time Stamps

Device time stamps are specified as part of a header definition, using the variables devts. There are two ways to specify a time stamp format:

- Standard predefined time stamp format
- Time stamps can be defined by a user-specified format string

Standard Predefined Time Stamps

There are two predefined time stamps:

- MDTS (Month, Day, TimeStamp)
- MDYTS (Month, Day, Year, TimeStamp)

The following are the standard predefined time stamp components.

Component	Definition
Mmm	Abbreviation for month of the year (case sensitive). Valid values are Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, and Dec.
Dd	Day of the month. Valid values are 1 through 31.
Hh	Hour of the day. Valid values are 00 through 23.

Component	Definition
Mm	Minute of the hour. Valid values are 00 through 59.
Ss	Second of the minute. Valid values are 00 through 59.
Yyyy	Year. Valid values are 1970 through current year.

MDTS (Month, Day, TimeStamp) predefined time stamp formats. The MDTS (dmonth, dday, dtime) format is Mmm dd hh:mm:ss.

The following is an example of MDTS in a log (device time stamp is in bold font):

Aug 18 12:48:30 [120.2.40.1] **OCT 31 15:59:05** 1/1 149 VRRP-4: Master is 10.20.30.40, VRID:1

The header definition for this log message is:

<HEADER id1="0001" id2="0001"
devts="MDTS(dmondth,dday,dtime)"
content="<dmonth> <dday> <dtime> <slot>/<port> <msgcount>
<messageid>-<level>: <!payload>" />

The variable devts="MDTS(dmondth,dday,dtime)" identifies that a device time stamp exists in the header, and the time stamp format is MDTS. The function's input <args> must be defined in the header content string.

MDYTS (Month, Day, Year TimeStamp) predefined time stamp format. The MDYTS (Month, Day Year Timestamp) format is Mmm dd yyyy hh:mm:ss.

The following is an example of MDYTS in a log (device time stamp is in bold font):

Aug 10 15:45:38 [10.10.20.1] **Aug 10 2004 15:44:51**: %PIX-5-304001: 10.10.20.128 Accessed URL 205.188.135.105:/redirects/inclient/AIM_UAC.adp?magic=93167109&width=120&height=90

The header definition for this log message is:

HEADER id1="0003" id2="0003"
devts="MDYTS(month,day,year,time)"
content="<month> <day> <year> <time> %PIX-<level>-<messageid>:
<!payload>" />

The variable devts="MDYTS(month,day,year,time)" identifies that a device time stamp exists in the header, and the time stamp format is MDYTS. The function's input <args> must be defined in the header content string.

Non-Standard Time Stamps

Non-standard time stamps are defined by a user-specified time stamp format string. The format string contains format codes, which describe to UDS how to process the time stamp string.

The following are the format codes for specifying a time stamp layout.

ASCII Date Component	Formatting Code	Example
Full Month Name	%R	January, JANUARY
Abbreviated Month Name	%B	Jan, JAN
Numeric Month	%M	01 – 12
Numeric Month Day	%D	01 – 31
Hour (24 hour period)	%H	00 – 23
Hour (12 hour period)	%I	00 – 11
AM/PM (Needed by %I)	%P	AM or PM
A.M./P.M. (Needed with %I)	%Q	A.M. or P.M.
Minutes	%T	00-59
Seconds	%S	00 – 59
Year (this century)	%Y	00 – 99
Year	%W	0000 – 9999
Julian Day	%J	001-364
%%	ASCII Percent	

The following log message contains a non-standard format device time stamp (time stamp is in bold font):

Aug 18 12:51:04 [110.110.7.1] id=firewall sn=004010103026 time="**2002-01-01 14:24:46**" fw=10.10.100.1 pri=6 c=1024 m=4 SonicWALL activated

A user-specified time stamp format for this log message is:

```
<HEADER id1="0001" id2="0001"
devts="'TS('%W-%M-%D %H:%T:%S',yearMonthDay,time)"
content="id=<id> sn=<sn> time="<yearMonthDay> <time>" fw=<fw> pri=<pri>
c=<c> m=<messageid> <!payload>" />
```

The TS function's input <args> must be defined in the header content string. The TS function supports up to 10 input <args>.

The prototype for the TS function is TS (formatString, var1, …). The ellipsis (…) indicates that the TS function supports a variable number of input arguments. This allows the time stamp to be spread over an unknown number of <args>. In the example, the time stamp is spread over two input <args>.

Building Headers That Use Device Time Stamps

Headers are contained within Word Action Nodes (WANodes) and Conditional Word Action Nodes (CWANodes). Four CWANode elements support time stamps:

- devTsXmlStr—devts string retrieved from the message XML file.
- devTsArgc—the number of arguments in devTsXmlStr.
- devTsArgv—an array of pointers, which point to arguments in devTsXmlStr.
- devTsMethod—a method pointer, which points to DeviceIParse method to construct time stamp.

```
// Conditional Word Action Node
class CWANode : public WANode
{
public:
   int typeCondition;
   CRegExp *cRegExp;
   CValueMap *cValueMap;
   bool(DeviceIParse::* conditionalMethod)(ConditionParm&);
   //Device Timestamp Attributes.
   char* devTsXmlStr;// XML Header devts string
   int devTsArgc;// Device timestamp argc
   char** devTsArgv;// Device timestamp argv
   //Method to construct device timestamp
   bool(DeviceIParse::* devTsMethod)
      (Kstring* buffer, const int argc, const char** argv, Kstring* target);
   CWANode(char* id) : WANode(id), conditionalMethod(0), typeCondition(0),
      cRegExp(NULL), cValueMap(NULL), devTsArgc(0), devTsArgv(NULL),
      devTsMethod(0) {}
   ~CWANode();
};
```

Device Initialization

When devices are loaded and initialized, UDS builds a header called Finite State Machine (FSM). During this time, CWANode classes are allocated and populated. If a Header has a device time stamp specification, all initialization to support the time stamps takes place.

The following table describes how time stamp oriented attributes are initialized.

Attribute	Initialization Details
devTsXmlStr	Populated with the time stamp in the XML Header devts string.
devTsXmlStr	Scanned for the type of device time stamp to be generated: MDTS, MYDTS, or TS. Based upon this scan, the attribute devTsMethod is set to an appropriate DeviceIParse instance method. These methods are: DeviceIParse::constructDevTsMYTS(*Kstring* buffer, const int argc, const char** argv, char* target*); DeviceIParse::constructDevTsMYDTS(*Kstring* buffer, const int argc, const char** argv, char* target*); DeviceIParse::constructDevTsMS(*Kstring* buffer, const int argc, const char** argv, char* target*);
devTsXmlStr	Parsed up into arguments. Arguments exist between the open and close parentheses in the devTsXmlStr attribute. For example, in devTsXmlStr=MYDTS(dmondth,dday,dtime):, there are three arguments in this time stamp definition. The attribute devTsArgc would be set to 3, and devTsArgc would be set to point into devTsXmlStr. Attribute devTsArgv would contain the following arguments: • devTsArgv[0] = “dmonth” • devTsArgv[1] = “dday” • devTsArgv[2] = “dtime” • devTsArgv[3] = NULL

Determining if a Message Contains a Device Time Stamp

At runtime, as messages are processed, XML Header Definitions are compared to incoming log messages. When a match is found between an XML Header’s content string and an incoming message, the XML Header Definition is used to process the message.

The DeviceIParse::parseHeaderData() method performs this comparison. When a Header /LogMessage match is found, the Header’s CWANode instance is checked to see if CWANode::devTsMethod is non-NULL. If it is non-NULL, this indicates that the message contains a device time stamp, and the DeviceIParse attribute is set to point to the CWANode instance containing the device timestamp definition.

The class DeviceIParse attribute and assessor methods associated with device time stamps are:

- CWANode* devTsCWANodeP;
- Bool devTsExists() { return (devTsCWANodeP) ? true : false; };
- CWANode* getDevTsDef() { return devTsCWANodeP; };

Check if Time Stamps Should Be Generated

To determine if device time stamps are to be processed, the system checks two conditions:

- Did the active driving thread tell the Parser to process device time stamps?
- Does the parsed message contain a device time stamp?

Generating Device Time Stamps

Time stamps are generated by firing the DeviceIParse instance method pointed to by CWANode::devTsMethod. No initialization or string copies are allowed in these methods. These actions would simply construct a time stamp.

The CGenericDevice has a DeviceIParse instance. This instance parses the message, and this instance fires the method pointed to by CWANode::devTsMethod. This method's execution generates a device time stamp.

bool(DeviceIParse::* devTsMethod)
(*Kstring* buffer, const int argc, const char** argv, char* target*);

The following table list the arguments for the devTsMethod method pointer.

Argument	Description
Kstring* buffer	Contains the log message. This argument is provided by the CGenericDevice instance.
const int argc	Number of arguments contained in argv. argc is provided by CWANode::argc. This value is set at initialization time.
const char** argv	Time stamp generation arguments—substrings of the XML Header devts string. This value is set at initialization time.
Kstringr* target	A buffer to contain the constructed time stamp string. This buffer is provided by the CGenericDevice instance. This buffer needs to be big enough to contain the constructed timestring.

The format of the generated time stamp in the target buffer is the same regardless of the devTsMethod fired. If a time stamp is successfully generated, the contents of the target buffer are loaded into instance attribute CCommonLStore::m_szStamp.

Messages

A message is made up of fixed and optional variables. All variables are delimited from one another, usually by spaces.

Here is an example of the breakdown of the Cisco PIX message 305012:

<MESSAGE level="6" parse="1" parsedefvalue="1" tableid="6" bucket=”x,y,z” id1="305012:00" id2="305012" content="Teardown {dynamic | static} {TCP | UDP | ICMP} translation from <finterface>:<faddr>/<fport> to <linterface>:<laddr>/<lport> duration <duration>" />

The following table describes the fields in the message.

Variable	Data
level	6
parse	1
parsedefvalue	1
tableid	6
bucket	x, y, and z summarizations
id1	305012:00
finterface	inside
faddr	192.168.1.24
fport	14848
linterface	dmz
laddr	192.168.5.254
lport	6159
duration	0:00:31

Message Order

The message order is very important for precedence. The system reads the message IDs in the order that they appear in the file. It does not sort them by message number.

For example, here are two messages:

A B C

A B C D

The message A B C D should appear first in the list. If not, the system reviews the first message in the list, A B C, and if the message matches, the system does not go any further into the list. The correct message order is:

A B C D

A B C

Data Reduction

Reduce the number of lines wherever possible. Use a pipe (|) to separate options for an element inside square brackets ([]) and braces ({}).

For example, for the message X A B C, you could have the three lines or reduce them to one line.

Three lines	X A <sport> X B <sport> X C <sport>
Reduced to one line	X {A|B|C} <sports>

For example, nine NetScreen device messages can be reduced to one line.

Nine lines	HA state of the local NetScreen device has changed from master to master HA state of the local NetScreen device has changed from master to backup HA state of the local NetScreen device has changed from master to init HA state of the local NetScreen device has changed from backup to master HA state of the local NetScreen device has changed from backup to backup HA state of the local NetScreen device has changed from backup to init HA state of the local NetScreen device has changed from init to master HA state of the local NetScreen device has changed from init to backup HA state of the local NetScreen device has changed from init to init
Reduced to one line	HA state of the local NetScreen device has changed from { master | backup | init } to { master | backup | init }

Conditional Variables

Conditional variables are:

- Value maps
- XML system functions
- Regular expressions.

The format for all types of conditional expressions is:

<targetParmName:*conditionalExpressionName(CSV parameter list)>

Value Map

Value maps, sometimes known internally as table lookups, provide the ability to substitute a DML <parm> value for an alternate value.

Value maps require three map definitions:

- name—handle used to reference the map in XML device definitions
- defaultvalue—value associated with any key that is not found in the map
- keyvaluepairs—set of hash key and associated value pairs

A value map has the following access syntax:

<*parm:target:valuemap>

The following table describes these elements.

Element	Description
*parm	Content of the parameter, to be used as key in the value map.
target	Location into which to place the value of the value map.
valuemap	Handle name of the value map.

The following is a sample value map:

```
<VALUEMAP
name="protocolNames"
default="PPP"
keyvaluepairs="28-500='PPP'|533='SMDS'|540='FRAMERELAY'|1723='ATM'" />
```

The following are sample usages of value maps. Notice that all UDS conditional expressions have the same usage syntax.

```
<SUMDATA
bucket="NIC_B_FW_ADDR_ACCOUNTING"
key="*protocolNames(lport)"
fields="1" />
```

```
<MESSAGE
category="6" level="4" parse="1" parsedefvalue="1" tableid="9"
id1="106023" id2="106023" content="Deny <protocol> src
<finterface:*protocolNames(lport)>:<faddr>/<fport> dst <linterface>:<laddr>/<lport> by access-group
"<listname>"" />
```

The following is an example of a value map of port to well-known name, <*sport:port:knownports>. If the content of <sport> was 1512, the string wins would be placed into <port>.

```
<VALUEMAP
   name="knownports"
   defaultvalue=”unknown”
   keyvaluepairs="514=’syslog’ | 533='NETWALL' | 540='uucp' | 543='klogin' |
   544='kshell' | 550='new-rwho' |556='remotefs' | 560='rmonitor' |
   561='monitor' | 636='ldaps' |666='doom' | 749='kerberos-adm' |
   749='kerberos-adm' | 750='kerberos-iv' | 1109='kpop' | 1167='phone' |
   1433='ms-sql-s' | 1433='ms-sql-s' | 1434='ms-sql-m' | 1434='ms-sql-m' |
   1512='wins' | 1512='wins' | 1524='ingreslock' | 1701='l2tp' |
   1723='pptp' | 1812='radius' | 1813='radacct' | 2049='nfsd' |
   2053='knetd' | 9535='man'"/>
```

The following is an example of a value map for a port-to-protocol type. If port=<dport> and dport = 514, protocol =UDP. The parameter <protocol> would be populated by the statement <*dport:protocol:port2protocol>.

```
<VALUEMAP
   name="port2protocol"
   defaultvalue=”unknown”
   keyvaluepairs="500='udp' | 512='udp' | 513='udp' | 514='udp' | 517='udp' |
```

518='udp' | 520='udp' | 525='udp' | 533='udp' | 550='udp' | 560='udp' |
561='udp' | 666='udp' | 749='udp' | 750='udp' | 1167='udp' | 1433='udp' |
1434='udp' | 1512='udp' | 1812='udp' | 1813='udp' | 2049='udp'" />

XML System Functions

XML system functions currently supported are functions with MIN, MAX, SUM, CAT, and so forth. These functions can be used in message or bucket definitions.

The following is an example XML function within a message. The value of <fport> and <lport> are added together: <lport> = <lport> + <fport>.

<MESSAGE
category="6"
level="4"
parse="1"
parsedefvalue="1"
tableid="9"
id1="106023" id2="106023"
content="Deny <protocol> src <finterface>:<faddr>/<fport:*portType(fport)> dst
<linterface:*lifName(finterface)>:<laddr>/<lport:*SUM(lport,fport)> by access-group
"<listname>"" />
Here is an example within a message definition:
<SUMDATA
bucket="NIC_B_FW_PORT_ACCOUNTING"
key="*MIN(lport,fport)"
fields="1" />

All conditional functions are denoted by an asterisk (*****).

Input Parameter Value Operator ~

Conditional expressions and system functions take input parameters. These input parameter values are by default the Pass 1 value of the parameter. For details, see "Data Parsing" on page 19.

The Parameter Value Operator allows Pass 2 parameter values to be referenced, when used as an input to a conditional or system function.

The following is the general format of the input parameter value operator:

<@targetParm:*valueMap(~parm)>

Regular Expressions

UDS supports regular expressions. The standard regex, lex, and java meta characters are supported. The following table lists some of the supported meta characters.

Character	Definition
^	beginning of line
$	end of line
.	any character

Character	Definition
[ ]	character range
\	escape for meta characters, so it can be a literal character
?	match preceding item 0 or 1 times
+	match preceding item 1 or more times
*	match preceding item 0 or more times

The following is an example of a regular expression:

If port=<dport>, and dport is greater than 500 and less than 600, set protocol =”UDP”:

```
<REGEX
   name="portType"
   parms="parm1"
   default="unknown"
   expr="<parm1>(‘5[0-9][1-9]’) ? ‘UDP’"
/>
```

The parameter <protocol> is populated by the following statement:

```
<@protocol:<*portType>(<dport>)>
```

Null Regular Expression Substitution String

When a null regular expression substitution string is provided, the regular expression uses the input parameter as a substitution string. For example:

```
<REGX
   name="classCnetwork"
   parms="parm1"
   default="$NONE"
   expr="<parm1>(192.*) ? ""
/>
<SUMDATA
   bucket="NIC_B_FW_ADDR_ACCOUNTING"
   key="*classCnetwork(faddr)"
   fields="1"
/>
<MESSAGE
   category="6" level="2" parse="1" parsedefvalue="1" tableid="9"
   summary="NIC_B_FW_ADDR_ACCOUNTING"
   id1="106006:01" id2="106006"
   content="Deny inbound <protocol> from <faddr>/<fport> to <laddr>/<lport>"
/>
```

In this example, if the regular expression pattern is not matched, no summary entry is made.

Keywords

All keywords begin with a dollar sign ($).

The following is an example of filtering conditional expressions using the keyword $NONE.

```
<REGX
   name="network208"
   parms="parm1"
   default="$NONE"
   expr="<parm1>(208.*) ? '208.0.0.0'"
/>
<SUMDATA
   bucket="NIC_B_FW_ADDR_ACCOUNTING"
   key="*network208(faddr)" fields="1"
/>
<MESSAGE
   category="6" level="2" parse="1" parsedefvalue="1" tableid="9"
   summary="NIC_B_FW_ADDR_ACCOUNTING"
   id1="106006:01" id2="106006"
   content="Deny inbound <protocol> from <faddr>/<fport> to <laddr>/<lport>"
/>
```

In this example, all IP addresses that begin with 208 are translated to 208.0.0.0. If the IP address does not begin with 208 because NONE is specified, no summary entry is made.

This feature allows you to categorize IP addresses under a single summary entry key and it provides filtering so that you can only count IP addresses that are of interest to you if you wish.

Parameter Names

All parameter names must begin with an alpha character. Parameters cannot begin with a numeric value, and parameter names cannot begin with an asterisk (*).

All items that begin with a number are considered to be numeric values.

System XML

The System XML file contains definitions that are common to all XML parser definitions.

Summaries

There are 16 predefined summaries in the application that can be used to minimize data and speed up the creation of reports. Some of these summary buckets are designed for specific devices or device classes, while new ones are added to handle any device data. The values that you can use in the summary are based on the summary class, or group, that is being used.

Summary Classes

There are three basic classes of summary in the application:

- Character Key Summary (commonly referred to as an Address Summary)
- Numeric Key Summary (commonly referred to as a Port Summary)
- Category Summary

Summary Class Parameters

The fields used to populate each summary are slightly different, although they all have the same basic structure.

Each message that is inserted into a summary table has three parameters that are automatically included in each summary that allow you to separate data for each summary on a per-device basis. The three parameters are:

- Table/Summary Number
- Device Address
- Device Type

The following are the remaining parameters that can be used for each summary class.

Summary Class	Parameters
Character Key Summary	Key SumType Counter 1 (ac1 – accumulative counter) Counter 2 (ac2 – accumulative counter)
Numeric Key Summary	Key SumType Counter 1 (ac1 – accumulative counter)
Category Summary	Key subkey SumType Counter 1 (ac1 – accumulative counter) Counter 2 (ac2 – accumulative counter) Counter 3 (ac3 – accumulative counter)

The following table contains definitions of the parameters.

Parameter	Definition
Key	Item for which all the data is summarized, for example, IP address. While the Character Key summaries are most commonly used for IP addresses, they can be used for any string data up to 64 characters long, such as Interface Name, Windows Computer Name, and Username. The Numeric Keys can only be a number, such as a port.
subkey	(Only used in Category Summaries) A character string to further define a key value for summary, for example, Windows Log filename and Message ID.
SumType	A numeric value that separates data into logical groups. For example, all addresses in the summary with a Type = 1 are connection based messages.
Counter	Any numeric data that can be incremented for each entry that matches the Device IP, Key, and Type values. The most common usage for these are bytes and duration, but they can be used for any numeric value, such as packets.

Summary Buckets

Of the sixteen predefined summaries, ten are designed to be used specifically for summaries of firewall and router device classes. The bucket names are designed to help identify the specific summaries.

The following table describes the buckets for each summary class.

Device Class	Bucket	Key Type	Name
Firewall	NIC_B_FW_BYTES_ACCOUNTING	Character	Firewall Accounting Byte Summary
	NIC_B_FW_ADDR_ACCOUNTING	Character	Firewall Accounting Address Summary
	NIC_B_FW_PORT_ACCOUNTING	Numeric	Firewall Accounting Port Summary
	NIC_B_FW_HTTP_REQUESTS	Character	Firewall URL Requests Summary

Device Class	Bucket	Key Type	Name
Routers	NIC_B_RTR_PORT_ACCOUNTING	Numeric	Router Accounting Port Summary
	NIC_B_RTR_ADDRESSS_ACCOUNTING	Character	Router Accounting Address Summary
	NIC_B_RTR_FW_BYTES	Character	Router FW Bytes Summary
	NIC_B_RTR_ADDRESS_SECURITY	Character	Router Security Address Summary
	NIC_B_RTR_PORT_SECURITY	Numeric	Router Security Port Summary
	NIC_B_RTR_FW_SESSION	Character	Router FW Session Summary
All devices (generic device data)	NIC_B_COUNTS	Category	Sumcount
	NIC_B_WINDOWS	Character	Windows Summary
	NIC_B_CATEGORIES	Category	Category Count Summary
	NIC_B_ADDRESS_ACCOUNTING	Character	Accounting Address Summary
	NIC_B_PORT_ACCOUNTING	Numeric	Accounting Port Summary
	NIC_B_URL_REQUESTS	Character	URL Requests Summary

The data that the summary buckets can hold is also based on the class.

Note: While the input of this summary class allows a subkey, it will only be output in the Category Count summary.

Here are the available values, with examples, that can be used in each class and set on a global basis or on an individual message basis.

Character key summary example:

Key,	SumType,	Counter1 (ac1),	Counter2 (ac2)
address,	connection,	bytes,	duration
10.10.30.14,	1,	100,	100

Numeric key summary example:

Key,	SumType,	Counter1 (ac1)
port,	permit,	packets
80,	1,	20

Category summary example:

Key,	SubKey,	SumType,	Counter1 (ac1),	Counter2 (ac2),	Counter3 (ac3)
hostname,	host class,	count,	sent bytes,	rec bytes	
www.rsa.com,	web,	1,	100,	0	

Using Summaries

The UDS System XML defines NIC buckets that are available to UDS. This file is located in **enVision/etc/devices/globaldef.xml**.

Here are the values in a bucket definition:

- Bucket name—NIC supported bucket name.
- Summary type—values other than zero have device-specific meanings.
- ref, reference, (table/summary ID).
- fields—valid values for **ac1** and **ac2**.

Here are the two summary structures:

- sumtypelist
- fieldslist

sumtypelist

The sumtypelist structure defines the number used in the **SumType** field. Use sumtypelist structures to define the data set. These values are defined in the **global.xml**.

For example, the following definition shows the current values used in the Firewall Accounting Address Summary (NIC_B_FW_ADDR_ACCOUNTING).

sumtypelist="default=0;
connection=1;
denied_in=2;
denied_out=3;"
The values are defined as:
1 = Connection Message
2 = Denied Incoming Connection
3 = Denied Outgoing Connection

The sumtype value is established in one of three ways:

- You can assign sumtype a constant value, for example:
  summary="NIC_B_FW_ADDR_ACCOUNTING;fields=0,0;sumtype=42”
- You can make sumtype reference a conditional statement, for example:
  summary="NIC_B_FW_ADDR_ACCOUNTING;fields=0,0;sumtype=
  *dir2SumType(inout);”
- If not explicitly defined in a message’s summary definition, the default bucket sumtype value is used, which is defined in **globaldef.xml**, for example:
  summary="NIC_B_FW_ADDR_ACCOUNTING;fields=0,0;”

  In this case the NIC_B_FW_ADDR_ACCOUNTING sumtype value is the default, which is defined as 0 (from **globaldef.xml**):

  NIC_B_FW_ADDR_ACCOUNTING:sumtype=default,ref=8,fields=
  ac1,ac2;

fieldslist

The fieldslist structure defines the default value used when a parameter is inserted into the summary. In most cases, the ac1 (Counter 1) can be defined in the XML as bytes. If a message does not specify a different value, the bytes variable is used whenever a summary calls for a value that includes ac1.

```
fieldslist="ac1=bytes;
  ac2=duration;
  ac3=1;"
```

The entire list of predefined buckets is included in global.xml. Each bucket defines the default SumType, internal reference number (ref), and the fields used in the summary (fields). For example:

NIC_B_RTR_PORT_ACCOUNTING: sumtype=default, ref=1, fields=ac1;

The following table contains details of each bucket.

Bucket Name	SumType	Ref	Fields Used in Summary
NIC_B_RTR_PORT_ACCOUNTING	default	1	ac1
NIC_B_RTR_ADDRESSS_ACCOUNTING	default	2	ac1, ac2
NIC_B_RTR_FW_BYTES	sdefault	3	ac1, ac2
NIC_B_RTR_ADDRESS_SECURITY	default	4	ac1, ac2
NIC_B_RTR_PORT_SECURITY	default	5	ac1
NIC_B_RTR_FW_SESSION	default	6	ac1, ac2
NIC_B_FW_BYTES_ACCOUNTING	default	7	ac1, ac2
NIC_B_FW_ADDR_ACCOUNTING	default	8	ac1, ac2
NIC_B_FW_PORT_ACCOUNTING	default	9	ac1

Bucket Name	SumType	Ref	Fields Used in Summary
NIC_B_FW_HTTP_REQUESTS	default	10	ac1, ac2
NIC_B_DEVICES	default	38	ac1, ac2
NIC_B_WINDOWS	default	42	ac1, ac2
NIC_B_CATEGORIES	default	43	ac3, ac1, ac2
NIC_B_ADDRESS_ACCOUNTING	default	59	ac1, ac2
NIC_B_PORT_ACCOUNTING	default	60	ac1
NIC_B_URL_REQUESTS	default	61	ac1, ac2

The buckets used in a specific device must be defined further in the **device.xml**. For example:

```
<SUMDATA
bucket="NIC_B_FW_ADDR_ACCOUNTING"
key="laddr" />

<SUMDATA
bucket="NIC_B_FW_PORT_ACCOUNTING"
key="lport" />
```

Summary bucket example 1

This is an example of how to assign a message to a summary bucket, taken from the Cisco PIX UDS XML.

Choose which summary you want to use from the ones defined in the **globaldef.xml** file.

```
sumtypelist="default=0;
ciscopix_Conn=1;
ciscopix_denyedIncommingConn=2;
ciscopix_denyedOutgoingConn=3;
firewall=9;
hub=3;
router=1;"
fieldslist="ac1=1; ac2=0; ac3=0;"
nicbuckets="NIC_B_RTR_PORT_ACCOUNTING:sumtype=default,ref=1,fields=ac1;
NIC_B_RTR_ADDRESSS_ACCOUNTING:sumtype=default,ref=2,fields=ac1,ac2;
NIC_B_RTR_FW_BYTES:sumtype=default,ref=3,fields=ac1,ac2;
NIC_B_RTR_ADDRESS_SECURITY:sumtype=default,ref=4,fields=ac1,ac2;
NIC_B_PORT_SECURITY:sumtype=default,ref=5,fields=ac1;
NIC_B_ROUTER_FW_SESSION:sumtype=default,ref=6,fields=ac1,ac2;
NIC_B_FW_BYTES_ACCOUNTING:sumtype=default,ref=7,fields=ac1,ac2;
NIC_B_FW_ADDR_ACCOUNTING:sumtype=default,ref=8,fields=ac1,ac2;
In the top of the device’s XML define the bucket you want to use.
<SUMDATA
bucket="NIC_B_FW_ADDR_ACCOUNTING"
key="laddr"
fields="bytes,0" />
```

Include the following tags for the message that you want to add to this bucket:

```
<MESSAGE
category="25"
eventcategory="1803010000"
niccategory="18"
parse="1"
parsedefvalue="1"
tableid="9"
id1="106001"
id2="106001"
summary="NIC_B_FW_ADDR_ACCOUNTING;fields=0,0;
   sumtype=ciscopix_denyedOutgoingConn;
   |NIC_B_FW_PORT_ACCOUNTING;
   sumtype=denyedIncommingConn;|NIC_B_DEVICES;"
content="&lt;@inout:*DIRCHK(faddr)&gt;Inbound &lt;protocol&gt;
   connection denied from &lt;faddr&gt;/&lt;fport&gt; to
   &lt;laddr&gt;/&lt;lport&gt;
   flags &lt;TCP_flags&gt; on interface
   &lt;interface&gt;&lt;@ntype:9&gt;&lt;
   @ntype:*ifSecurityDirOutBound(inout)&gt;" />
```

There are multiple handling flags associated with this message. For this example, only the FW_ADDR_ACCOUNTING one is discussed. This tells enVision on which fields to perform the accounting summary. These buckets create the summary data found in the Address Accounting Summary table so that you can report on Bytes by Device Address, and so forth.

Summary bucket example 2

Here is a summary bucket defined for address accounting:

- The key for the bucket is the <laddr> parameter (as defined in the SumData tag in the previous section).
- The SumType value is for a “connection” message.

The accumulative count ac1 and ac2 values are not defined, Therefore, ac1 value is the value in the bytes parameter, and the ac2 value is the value in the duration parameter, as defined in the System XML in the previous section.

```
<MESSAGE
level="6"
parse="1"
parsedefvalue="1"
tableid="12"
id1="302014:01"
id2="302014"
summary="NIC_B_FW_ADDR_ACCOUNTING;sumtype=connection"
content="Teardown TCP connection <accountid> for
   <finterface>:<faddr>/<fport> to
   <linterface>:<laddr>/<lport> duration <duration> bytes <bytes>" />
```

The following message creates an entry into the Firewall Accounting Address Summary:

Sep 15 12:00:00 [10.10.20.1] %PIX-6-302014: Teardown TCP connection 2000 for outside:202.202.202.202/80 to inside:192.168.1.129/1520 duration 0:01:00 bytes 100 (user)

The values used for the insert are as follows:

(**NIC_B_FW_ADDR_ACCOUNTING**)
Device Type = 0 – Added by default. Cannot change.
Device Address = 10.10.20.1 – Added by default. Cannot change.
Key (laddr)= 192.168.1.129
SumType (connection)= 1 – Defined in the sumtypelist.
ac1 (bytes)= 100
ac2 (duration)= 60 – One minute in seconds.

Overriding the Default Summary Key

You can define the key used for the summary entry on a single message by overriding the key tag.

The following is an example, using <faddr> for the key, instead of the default <laddr>.

<MESSAGE
level="6"
parse="1"
parsedefvalue="1"
tableid="12"
id1="302014:01"
id2="302014"
summary="NIC_B_FW_ADDR_ACCOUNTING;key=faddr;sumtype=connection"
content="Teardown TCP connection <accountid> for <finterface>:<faddr>/<fport> to <linterface>:<laddr>/<lport> duration <duration> bytes <bytes>" />

The following message creates an entry into the Firewall Accounting Address Summary:

Sep 15 12:00:00 [10.10.20.1] %PIX-6-302014: Teardown TCP connection 2000 for outside:202.202.202.202/80 to inside:192.168.1.129/1520 duration 0:01:00 bytes 100 (user)

The values used for the insert are as follows:

(**NIC_B_FW_ADDR_ACCOUNTING**)
Device Type = 0 – Added by default. Cannot change.
Device Address = 10.10.20.1 – Added by default. Cannot change.
Key (faddr)= 202.202.202.202
SumType (connection)= 1 – Defined in the sumtypelist.
ac1 (bytes)= 100
ac2 (duration)= 60 – One minute in seconds.

Accessing Multiple Buckets within a Message

You can generate multiple summaries on an individual message. To add data to more than one summary, include all bucket names, separated by a pipe (|). For example:

<MESSAGE
level="6"
parse="1"
parsedefvalue="1"
tableid="12"
id1="302014:01"
id2="302014"
summary="NIC_B_FW_ADDR_ACCOUNTING;sumtype=connection|
NIC_B_FW_PORT_ACCOUNTING;sumtype=connection "

content="Teardown TCP connection <accountid> for <finterface>:<faddr>/<fport> to <linterface>:<laddr>/<lport> duration <duration> bytes <bytes>" />

The following message creates an entry into the Firewall Accounting Address Summary AND the Firewall Accounting Port Summary:

Sep 15 12:00:00 [10.10.20.1] %PIX-6-302014: Teardown TCP connection 2000 for outside:202.202.202.202/80 to inside:192.168.1.129/1520 duration 0:01:00 bytes 100 (user)

The values used for the insert are as follows:

(**NIC_B_FW_ADDR_ACCOUNTING**)
Device Type = 0 – Added by default. Cannot change.
Device Address = 10.10.20.1 – Added by default. Cannot change.
Key (laddr)= 192.168.1.129
SumType (connection)= 1 – Defined in the sumtypelist.
ac1 (bytes)= 100
ac2 (duration)= 60 – One minute in seconds.

(**NIC_B_FW_PORT_ACCOUNTING**)
Device Type = 0 – Added by default. Cannot change.
Device Address = 10.10.20.1 – Added by default. Cannot change.
Key (lport)= 1520
SumType (connection)= 1 – Defined in the sumtypelist.
ac1 (bytes)= 100
ac2 – Not available in this summary.

Denied Incoming Messages

Here is an example of inserting a “denied incoming” message into the Firewall Accounting Address Summary. To do so, change the SumType to a “denied_in” (Denied Incoming) message.

<MESSAGE
category="**6**"
level="**3**" \
parse="**1**"
parsedefvalue="**1**"
tableid="**9**"
id1="**106010**"
id2="**106010**"
summary="**NIC_B_FW_ADDR_ACCOUNTING;sumtype=denied_in**"
content="**Deny inbound <protocol> src outside: <faddr> dst inside: <laddr>**" />

The following message creates an entry into the Firewall Accounting Address Summary:

Sep 15 12:00:00 [10.10.20.1] %PIX-3-106010: Deny inbound TCP src outside:12.15.105.8 dst inside:192.168.1.8

The values used for the inserts are as follows:

(**NIC_B_FW_PORT_ACCOUNTING**)
Device Type = 0 – Added by default. Cannot change.
Device Address = 10.10.20.1 – Added by default. Cannot change.
Key (laddr)= 192.168.1.8
SumType (connection)= 2 – Defined in the sumtypelist.
ac1 (bytes) = 0
ac2 (duration) = 0

The bytes and duration values are not defined in the above message, so they are inserted with a default value of 0.

XML Utility Functions

XML utility functions include:

- BYTES
- DUR
- CALC
- DIRCHK
- Flip
- HDR
- PARMVAL
- RMQ
- STRCAT
- SYSVAL
- URL
- UTC

BYTES

Use the BYTES function to convert bytes to normalized units. Assign a value of bytes to a <variable>.

BYTES (conversionType,inputParm)

The supported conversion types for BYTES are as follows.

Conversion type	Conversion Performed
$KB2B	Kilobytes to bytes
$MB2B	Megabytes to bytes
$GB2B	Gigabytes to bytes
$B2KB	Bytes to kilobytes
$B2MB	Bytes to megabytes
$B2GB	Bytes to gigabytes

Example: BYTES($B2KB, bytes)

Assume that the following log message is processed:

Jul 30 10:09:06 [10.10.20.128] %PIX-6-302002: Teardown TCP connection 253 faddr 12.15.105.10/54 gaddr 209.67.241.11/55 laddr 192.168.1.58/56 duration 1:00:05 bytes 10925 (TCP Reset-I)

Also assume the following XML message content string definition:

```
content="<@inout:*DIRCHK(faddr)>Teardown TCP connection <numer>
faddr <faddr>/<fport> gaddr <gaddr>/<port> laddr <laddr>/<lport> duration <duration> bytes <bytes>
(<username1>)<@ntype:1>
<@level:*SYSVAL($LEVEL)>
<@bytes:* BYTES($B2KB, bytes)>" />
```

The variable <bytes> is populated with 10925 from the log message content. The BYTES function converts 10925 bytes to 10 Kbytes.

DUR (Duration)

Use the DUR function to construct a time duration (DUR) value and assign it to a message variable. Duration is an interval of time normalized to seconds.

DUR(*msgSegment*, *formatString*, *inputParm* …)

The following table describes the parameters.

Parameter	Details
msgSegment	Defines where source inputParm variable resides: • $MSG—variables for date reside in message. • $HDR—variables for date reside in header.
formatString	Describes the format of the ASCII time contained within the inputParm. The formatString defines how the ASCII time is to be interpreted, so that a DUR (duration in seconds) can be generated. A formatString is composed of the following formatting codes:
inputParam	Input variable containing an ASCII text time. Either of the following: • DUR($HDR,' %H:%T:%S', completeDuration) • DUR($HDR,’ %H:%T:%S’, var1,var2,var3,var4...)

ASCII Date Component	Formatting Code	Example
Full Month Name	%R	January, JANUARY
Abbreviated Month Name	%B	Jan, JAN
Numeric Month	%M	01 – 12
Numeric Month(*variable width field)	%G	01 – 12
Numeric Month Day	%D	01 – 31
Numeric Month Day (*variable width field)	%F	01 – 31
Hour (24 hour period)	%H	00 – 23
Hour (12 hour period)	%I	00 – 12
Hours:Min:Sec	%Z	01:20:52
AM/PM (Needed by %I)	%P	AM or PM
A.M./P.M. (Needed with %I)	%Q	A.M. or P.M.
Minutes	%T	00 – 59
Minutes (*variable width field)	%U	00 – 59
Seconds	%S	00 – 59
Year (this century)	%Y	00 – 99
Year	%W	0000 – 9999
Julian Day	%J	001 – 364
%%	ASCII Percent	

(*the format code determines the size of the field for variable width fields)

Example: DUR($MSG,' %H:%T:%S',duration)

Assume that the following log message is processed:

Jul 30 10:09:06 [10.10.20.128] %PIX-6-302002: Teardown TCP connection 253 faddr 12.15.105.10/54 gaddr 209.67.241.11/55 laddr 192.168.1.58/56 duration 1:00:05 bytes 925 (TCP Reset-I)

Also assume the following XML message content string definition:

```
content="<@inout:*DIRCHK(faddr)>Teardown TCP connection <numer>
faddr <faddr>/<fport> gaddr <gaddr>/<port> laddr <laddr>/<lport> duration <duration> bytes <bytes>
(<username1>)<@ntype:1><@level:*SYSVAL($LEVEL)>
<@duration:*DUR($MSG,'%H:%T:%S',duration)>" />
```

The <duration> variable is initially populated with a value of 1:00:05 from the message content. The DUR function converts the <duration> value from 1:00:05 to 3605, which is equal to one hour and five seconds.

If a log message has a varying amount of space in between the time stamp variables, the DUR command automatically makes adjustments for these variations. If a field (such as hours above) has one digit instead of two digits, appropriate internal parsing adjustments are made to support that structure.

CALC (Calculate)

Use the CALC function to perform calculations. CALC supports the following operators.

Operator	Definition
+	Arithmetic addition
-	Arithmetic subtraction
*	Arithmetic multiplication
/	Arithmetic division
%	Arithmetic modulo
>>	Shift right
<<	Shift left

CALC supports no order of precedence. Values are evaluated from left to right. Parentheses are not supported.

The following is an example from Ciscopix, where the CALC function is used:

```
<MESSAGE category="6" level="3" parse="1" parsedefvalue="1" tableid="9" id1="106014"
id2="106014"
content="Deny inbound <protocol> src <idsinterface>: <faddr> dst <interface1>: <laddr>
(type <bytes>, code
<duration><@protocol:ICMP>)<@ntype:*CALC(bytes,*,100,+,duration)>" />
```

The following <ntype> is to set using the following calculation:

ntype = bytes * 100 + duration

The following expression can be directly represented using CALC:

<@ntype:*CALC(bytes,*,100,+,duration)>

DIRCHK (Direction Check)

You can use DIRCHK to:

- Set a variable to the direction of the current message and check it.
- Flip <parm> pairs, based on message direction.

Set a Variable to the Direction of the Current Message

Format: <@inout:*DIRCHK(faddr)>

The use of DIRCHK sets the variable inout to the message direction, either INBOUND or OUTBOUND. This is done by checking the IP address of the input parameter (faddr in this case) against a configured IP address table (ipaddr.tab) to determine whether the address is INSIDE or OUTSIDE of your network:

- If the address maps to an inbound ipaddr.tab entry, the DIRECTION is set to 0, indicating INBOUND.
- If the address maps to an outbound ipaddr.tab entry, the DIRECTION is set to 1, indicating OUTBOUND.

Flip <parm> Pairs Based on Message Direction

Many <parms> have a connotation of source or destination related to them, for example, faddr/laddr and fport/lport. For a number of NIC analysis algorithms (which generate queries and reports), it is important that the designation of source and destination be relative to the associated enterprise or AS (Autonomous Systems). A message is said to be INBOUND if it is coming in to the enterprise, and it is considered OUTBOUND if it is leaving the enterprise.

For messages where this is important, this XML function aligns a message's <parms> so that they match the Enterprise Relative Direction (ERD) of the message, as it relates to the enterprise or AS.

Example: For the following format, the input parameter faddr is checked against the ipaddr.tab to determine direction.

<@inout:*DIRCHK($OUTBOUND, faddr,laddr,...)

If the ERD of faddr matches the direction specified by parameter 1, the system flips the values of all <parm> pairs. This synchronizes the <parm> values to the message ERD.

Item	Description
parameter 1	Direction. In this case, faddr implies Foreign Address, which would be an external address. laddr implies the Local Address. If the result of the direction check indicates that faddr is actually a local address, the values of faddr and laddr need to be swapped. That would also apply to the values of the corresponding port variables, fport and lport.

Item	Description
parameter 2	Parameter to which the directional check is applied. If this parameter's direction does not match the requested direction, the message is aligned to requested direction, by flipping the specified <parm> sets.
parameter 3	Related to but directionally opposite the address associated with parameter 2.
Ellipsis (…)	Arbitrary number of conjugate <parm> sets.

Note: $OUT can be used as an alias for $OUTBOUND. $IN can be used as an alias for $INBOUND.

Example**:** If the direction (ERD) determined by DNS for parameter 2 matches the specified direction in parameter 1 then the system flips the values of the <parm> pairs.

<@inout:*DIRCHK($OUTBOUND, faddr,laddr,fport,lport,.....)>

This flip synchronizes the following:

- parameter 1—Direction. Normally the direction defaults to INBOUND because faddr occurs first. If the actual direction based on the DNS determination is OUTBOUND, the parameters are swapped.
- parameter 2—Parameter used for the DNS direction check (must be an IP address). This determines the actual message direction which is checked against parameter 1. If the 2 directions are the same, the parm pairs are swapped.

DIRCHK Examples

Below are three examples that show how to request that DIRCHK orient the variable pairs specified to the correct direction if faddr is not a foreign address.

Example 1:

<MESSAGE category="**6**" level="**4**" parse="**1**" parsedefvalue="**1**"
tableid="**9**" id1="**106023**" id2="**106023**"
summary="**NIC_B_FW_PORT_ACCOUNTING;fields=*sumPortAc(lport)**"
content="**Deny <protocol> src <finterface>:<faddr>/<fport:*MAX(lport,fport)> dst <linterface>:<laddr>/<lport:*MIN(lport,fport)> by access-group "<listname>"<@inout:*DIRCHK($OUTBOUND,faddr,laddr)>**" />

Example 2:

<MESSAGE category="**6**" level="**4**" parse="**1**" parsedefvalue="**1**"
tableid="**9**" id1="**106023**" id2="**106023**"
summary="**NIC_B_FW_PORT_ACCOUNTING;fields=*sumPortAc(lport)**"
content="**Deny <protocol> src <finterface>:<faddr>/<fport:*MAX(lport,fport)> dst <linterface>:<laddr>/<lport:*MIN(lport,fport)> by access-group "<listname>"<@inout:*DIRCHK($OUT,faddr,laddr,fport,lport)>**" />

Example 3:

<MESSAGE category="**6**" level="**4**" parse="**1**" parsedefvalue="**1**"
tableid="**9**" id1="**106023**" id2="**106023**"
summary="**NIC_B_FW_PORT_ACCOUNTING;fields=*sumPortAc(lport)**"

content="**Deny <protocol> src <finterface>:<faddr>/<fport:*MAX(lport,fport)> dst <linterface>:<laddr>/<lport:*MIN(lport,fport)> by access-group**

HDR (Header)

Use the header (HDR) function to reference header values in a message.

The following is an example of the HDR function:

```
<MESSAGE
eventcategory="1605000000"
niccategory="16"
parse="1"
id1="503001"
id2="503001"
content="Process number, Nbr <ip_address> on <string1> from <string2> to <string3>,
<string4><@msg:*RMQ($MSG)><@level:*HDR(level)>" />
```

In this example, the level parameter of the message is set as a value in the header parameter named level: <@level:*HDR(level)>.

The following is the definition of level in the header:

```
<HEADER
id1="0001"
id2="0001"
content="%PIX-<level>-<messageid>: <!payload>" />
```

In this example, the level <parm> of the message is set to the value of the HEADER's messageid: <@level:*HDR(messageid)>.

PARMVAL

Use the PARMVAL system function to assign the value of one parameter to another parameter's value.

The general format is:

@targetParm:*PARMVAL(INPUTPARAM, $MSG or $SGD)

Here is an example of PARMVAL:

<fport:*PARMVAL(lport)>

This statement assigns the value of <lport> to <fport>. If the value of <lport> is 23 then <fport> is assigned the value 23.

The reserved words $MSG and $SGD are used for internal operations:

- @msg:*PARMVAL($MSG) causes up to 512 characters of the syslog message to be copied to the internal variable "msg," which holds the syslog message contents.
- @sigcat:*PARMVAL($SGD) causes up to 256 characters of the syslog message to be copied to the internal variable "sigcat," which holds the Netscreen signature category.

RMQ (Remove Quotes)

Use the RMQ (Remove any Quotes) system function to remove quotes from the value of a a parameter.

The general format is:

@targetParm:*RMQ(INPUTPARM or $MSG)

Here is an example of RMQ:

<@msg:*RMQ($MSG)>

The keyword $MSG indicates the log message. This statement removes any quotes from the log message and places the resulting string into parameter <msg>.

STRCAT

Use the STRCAT (string concatenate) function to concatenate fields.

The general format is:

STRCAT (field,'delimiter',field)

Use STRCAT to create the complex message IDs used in Windows. The Windows message ID is comprised of three fields, joined by underscores. The format for this is:

STRCAT(msgIDPart1,'_',msgIDPart2,'_',msgIDPart3)

SYSVAL

Use the SYSVAL (system value) function to populate a parameter value with information that is outside of a **devicemsg.xml** document.

The general format is:

@targetParm:*SYSVAL($KEYWORD)

Defined SYSVAL keywords are:

- $MSGID,$ID1

  Use this function:

  – To move the Message ID1 value to the "message_id" field to override the default of ID2.

  – When ID2 contains ASCII text and a numeric representation of the message ID is desired.

- $LEVEL

  In the following example, the <level> parameter is set to the MESSAGE level="7". $LEVEL references the value of level within the XML MESSAGE element.

```
MESSAGE
category="2"
eventcategory="1603110000"
niccategory="16"
level="7"
id1="214001"
 id2="214001"
content="Terminating manager session from <IP_addr> on interface <int_name>. Reason:
incoming encrypted data (<number> bytes) longer than <upper_limit_number> bytes
<@msg:*RMQ($MSG)><@level:*SYSVAL($LEVEL)>" />
```

URL

Assign a URL substring to a <variable>.

URL(substring,inputParm)

URL supports the following substring types:

- $DOMAIN
- $ROOT
- $PAGE
- $QUERY

Example: Assign to variable <urldomain> the domain substring contained with in input parameter <urlstr>.

```
<MESSAGE     level="6"
category="10"
eventcategory="1401010000"
niccategory="14"
parse="1"
parsedefvalue="1"
sitetrack="1"
tableid="22"
id1="303002" id2="303002" summary="NIC_B_FW_HTTP_REQUESTS"
content="<username>@<laddr> <action> <faddr>:<urlstr><@ntype:*action2nType(action)>
<@urldomain:*URL(urlstr)> <@protocol:*action2Proto(action)>
<@level:*SYSVAL($LEVEL)>
. />
```

UTC

Use the UTC (Coordinated Universal Time, formerly known as Greenwich Mean Time) function to provide a facility to normalize the time coming from the payload of an event. Construct a UTC value, and assign it to a message variable.

UTC(msgSegment, formatString, inputParm …)

The following table describes the parameters.

Parameter	Details
msgSegment	Defines where the source inputParm variable resides: • $MSG—variables for date reside in message. • $HDR—variables for date reside in header.

Parameter	Details
formatString	Describes the format of the ASCII time contained within the inputParm. The formatString defines how the ASCII time is to be interpreted, so that a UTC time stamp can be generated. A formatString is composed of the following formatting codes:

ASCII Date Component	Formatting Code	Example
Full Month Name	%R	January, JANUARY
Abbreviated Month Name	%B	Jan, JAN
Numeric Month	%M	01 – 12
Numeric Month(*variable width field)	%G	01 – 12
Numeric Month Day	%D	01 – 31
Numeric Month Day (*variable width field)	%F	01 – 31
Hour (24 hour period)	%H	00 – 23
Hour (12 hour period)	%I	00 – 12
Hours:Min:Sec	%Z	01:20:52
AM/PM (Needed by %I)	%P	AM or PM
A.M./P.M. (Needed with %I)	%Q	A.M. or P.M.
Minutes	%T	00 – 59
Minutes (*variable width field)	%U	00 – 59
Seconds	%S	00 – 59
Year (this century)	%Y	00 – 99
Year	%W	0000 – 9999
Julian Day	%J	001 – 364
%%	ASCII Percent	

(*the format code determines the size of the field for variable width fields)

inputParm …	Input variable containing an ASCII text time. One of the following formats: • UTC($HDR,'%B %D %H:%T:%S', completeTimeStamp) • UTC($HDR,'%B %D %H:%T:%S', var1,var2,var3,var4...) If date string resides in multiple <variables>, these variables can be included as input to the UTC function

Example: UTC($HDR,'%B %D %H:%T:%S',month,day,timestamp)

```
<HEADER id1="0008" id2="0008"
content="<month> <day> <timestamp> <messageid>[<pid>]: <!payload>" />
<MESSAGE category="14" level="4" tableid="55" id1="CRON" id2="CRON"
niccategory="16" eventcategory="1605000000" content="<@msg:*PARMVAL($MSG)>( <username> ) CMD
( <command>
<@date:*UTC($HDR,'%B %D %H:%T:%S',month,day,timestamp)>"/>
```

If the following log message is processed:

Nov 12 19:00:00 CRON[13934]: (root) CMD (newsyslog)

The variables will have the following values:

- <month> will contain "Nov"
- <day> will contain "12"
- <timestamp> will contain 19:00.00

Notice that the format string has spaces between the month, day, and hour. The format string should always reflect the format of the date string in the log message (including spaces), if possible.

If a log message has a varying amount of space in between the time stamp variables, the UTC command automatically makes adjustments for these variations.

If a year value is available in the format string, this value will be used by the UTC command. If a year value is not available, as in this example, the year value associated with the receive time of the message is used.

XML Message Table IDs

Each RSA enVision table has an XML table ID. For example, the following XML Message Element defines **tableid=21**. This element indicates that this message goes into the Firewall System table.

```
<MESSAGE
   category="8"
   parse="1"
   parsedefvalue="1"
   tableid="21"
   id1="101003"
   id2="101003"
   content="(&lt;priority&gt;) Failover cable not connected
   (this unit)"
/>
```

Tables

The following table lists the table IDs for the RSA enVision tables.

Table Name	XML Table ID (tableid)	Supported Parameter Values
ACCESS_ACCOUNTING	27	<inout> <ntype> <msg> <username> <Caller-ID> <group> <rule> <duration> <service> <recv_bytes> <sbytes> <sportname> <priv-lvl> <protocol> <faddr> <fport> <laddr> <lport> <conn_id> <cmac> <smac> <vlan> <action> <lhost> <stamp>
ACCESS_SECURITY	24	<inout> <ntype> <msg> <shost> <product> <protocol> <list_name> <threshold> <username> <product>
ACCESS_SYSTEM	32	<inout> <ntype> <msg> <saddr> <shost> <product> <serial_number> <version> <release> <count> <result>
ACCESSCONTROL	30	<inout> <ntype> <msg> <event_type> <username> <connection_id> <sportname> <saddr> <reason>

Table Name	XML Table ID (tableid)	Supported Parameter Values
ACCOUNTING	12	<inout> <ntype> <msg> <duration> <bytes> <action> <direction> <rule> <protocol> <sbytes> <rbytes> <service> <icmptype> <icmpcode> <laddr> <lport> <finterface> <faddr> <fport> <username> <gaddr> <linterface> <accountid> <proto> <policy_id> <dst_zone> <src_zone> <vsys> <level> <misc_id> <reason> <data> <int_name> <description> <file>
ALERT_TRENDS	38	<inout> <ntype> <msg> <name> <category> <level> <rank> <weight> <apm> <change> <peak> <severity> <percent1> <percent2> <percent3> <percent4> <percent5> <trend> <prevscore> <score> <nav> <rate> <mpercent> <hpercent> <dpercent> <wpercent> <mrate> <hrate> <drate> <wrate>
AUDIT	33	<inout> <ntype> <msg> <auditdata>
AUTHENTICATION	3	<inout> <ntype> <msg> <interface> <result> <direction> <service> <protocol> <faddr> <fport> <laddr> <lport> <duration> <listname> <vsys> <level> <username>
BLOCKURL	36	<inout> <ntype> <msg> <url> <laddr> <faddr> <vsys> <level>
CALLDATA	45	<inout> <ntype> <msg> <handle_id> <service> <misc> <category> <dst_zone> <src_zone> <user_id> <addr> <mask> <rule> <reason> <result> <duration>
CONFIG_CHANGES	19	<inout> <ntype> <msg> <phost> <fhost> <username> <version> <project> <device> <result>
CONFIG_LEVEL	16	<inout> <ntype> <msg> <level>
CONFIG_SYSTEM	25	<inout> <ntype> <msg> <phost> <faddr> <fhost> <port> <version> <project> <resource> <filter> <mode>
CONFIG_USERACTIVITY	18	<inout> <ntype> <msg> <phost> <faddr> <fhost> <username> <action> <version> <group> <project> <profile>
CONTENTLEVEL	46	<inout> <ntype> <msg> <level>
CONTENTSYSTEM	47	<inout> <ntype> <msg> <filename> <reason> <directory> <result> <domain> <event_type> <url> <status> <laddr> <lport> <service>
FWSECURE	48	<inout> <ntype> <msg>
HTTP	22	<inout> <ntype> <msg> <sport> <action> <dport> <url> <protocol> <direction> <rule> <reason> <laddr> <faddr> <vsys> <level>
IDS	20	<inout> <ntype> <msg> <srcloc> <dstloc> <sigcat> <severity> <version> <hostId> <listnum> <faddr> <laddr> <sigid> <context> <sport> <dport> <sensor> <stamp> <protocol> <vsys> <level> <source> <saddr> <daddr> <product> <circuit> <detail>

Table Name	XML Table ID (tableid)	Supported Parameter Values
INTERFACE	49	<inout> <ntype> <msg>
IPS_ACCOUNTING	23	<inout> <ntype> <msg> <product> <duration> <protocol> <faddr> <fport> <laddr> <lport> <conn_id> <username> <cmac> <smac> <vlan> <rule> <action> <service> <lhost> <stamp>

Glossary

A-SRV

See Application Server.

ad hoc report

An unscheduled report that runs immediately.

ADB

See Asset Database.

administrator

A user responsible for setting up and maintaining the RSA enVision platform. An administrator has access to all enVision functions.

alert

An indication that an event, or a sequence of events, requires further investigation. The enVision platform sends alerts based on messages received under a configured set of circumstances such as filters. The administrator defines alerts for each view.

Alert History tool

The RSA enVision tool that is used to display alerts from the events database.

Alerts module

The RSA enVision module that provides tools to monitor, display, and configure alerts.

Analysis module

The RSA enVision module that provides tools to view, query, and analyze collected data.

appliance

The hardware on which RSA enVision software is deployed. See single appliance site and multiple appliance site.

Application Server (A-SRV)

The appliance or component of the RSA enVision platform that supports interactive users and runs the suite of enVision analysis tools. In a single appliance site, the Application Server (A-SRV) is a component of the enVision system. In a multiple appliance site, the A-SRV is installed on its own appliance. See single appliance site and multiple appliance site.

asset

A system, such as a host, software system, workstation, or device, that is within a network and makes up the enterprise environment.

Asset Database (ADB)

A unified view of assets created by merging data from supported vulnerability assessment (VA) tools and imported asset information in the asset tracking tools. The ADB provides security managers with insight into their operations.

attribute category

A group of categories defined by the RSA enVision platform for device and asset attributes. The nine categories are properties, location, organization, owner, physical, function, importance, vulnerability, and zone. Users can define custom categories.

bind report

A group of reports that can be scheduled to run as a single report.

collection

The process of collecting, analyzing, and storing logs from event sources. the RSA enVision platform stores the logs, with descriptive metadata, in the Log Smart Internet Protocol Database (IPDB).

Collector

The appliance or component of the RSA enVision platform that captures incoming events. In a single appliance site, the Collector is a component of the enVision system. In a multiple appliance site, the Collector is installed on its own appliance.

Common Storage Directory (CSD)

A single directory that contains the configuration and statistical information for data collected on a site. The Common Storage Directory (CSD) can be located on a single appliance site, on the Database Server of a multiple appliance site, or on the Remote Collector of a distributed system.

computer name

See node.

confidence level filtering

A filter defined by the administrator to determine if a supported intrusion detection system (IDS) or an intrusion prevention system (IPS) can be trusted for its truthfulness and applicability. The confidence level detects if a message from an IDS or an IPS should be considered an alert.

Configuration database (nic.db)

A repository that stores a user's configuration settings such as user information, permissions, and views.

correlation

A relationship between a set of events and a set of specific conditions.

D-SRV

See Database Server.

Database Server (D-SRV)

The appliance or component of the RSA enVision platform that manages access and retrieval of captured events. In a single appliance site, the Database Server (D-SRV) is a component of the enVision system. In a multiple appliance site, the D-SRV is installed on its own appliance. See single appliance site and multiple appliance site.

device

See event source.

device class

Identifies the classification of the event source. A device class provides a framework for organizing event sources by their general function.

device type (dtype)

An assigned internal name for an event source that is used by RSA enVision tools and utilities. The dtype value is displayed on the enVision interface, reports, and queries.

EA

See Enhanced Availability.

Enhanced Availability (EA)

A site with Enhanced Availability (EA) is a multiple appliance site where the Local Collector (LC) functionality runs on Cluster Appliances (CAs).

EPS

See events per second.

event category

System-defined or administrator-defined group of messages for alerting and reporting that is assigned across device classes.

Event Explorer

RSA enVision module that provides advanced tools for analysis of real-time and historical data. These tools allow users to sift through logged data and apply security forensics.

event source

An asset such as a physical device, software, or appliance that produces a message (log) and is configured to send the log to the RSA enVision platform. Event sources include firewalls, VPNs, antivirus software, operating systems, security platforms, routers, and switches.

events per second (EPS)

Events captured per second by the RSA enVision platform.

incident escalation

See task escalation.

incident management

See task triage.

IPDB

See LogSmart IPDB.

LC

See Local Collector.

Local Collector (LC)

A component of an RSA enVision multiple appliance site that captures incoming events. A multiple appliance site can have up to three Local Collectors (LCs). See multiple appliance site.

LogSmart IPDB

The LogSmart Internet Protocol Database (IPDB) stores internet protocol-based information, storing each source element in a separate container. Each log data message is identified by the IP address of the event source from which the message originated. The LogSmart IPDB maps this IP address to the originating event source and determines the format of the incoming message. The log message is the metadata that describes the event.

message category

A group of messages. Message categories are hierarchical, consisting of up to five levels: a NIC category, an alert category, and up to three levels of event category.

message variable

Defines a type of data that is extracted from message payloads. Message variables are useful when analyzing and reporting on data.

monitored device

A supported event source that has been configured to send event messages to the RSA enVision platform. The enVision platform collects and stores events from monitored devices.

multiple appliance site

An RSA enVision site in which each enVision component (Application, Collector, and Database) is on its own appliance.

NIC

The acronym used to label many essential RSA enVision components, services, and tools.

NIC database

See Configuration database (nic.db).

NIC domain

A group of multiple appliance sites that constitute an organization's entire deployment of the RSA enVision platform. One site acts as the NIC domain master site.

NIC message ID

A number that identifies a message. This number may or may not be the same as the vendor message ID.

NIC System device

Generates event messages to indicate the health and activity of the RSA enVision platform, such as disk space usage, current EPS, data retrieval statistics, and user activity messages.

NIC_View

Allows users to monitor the health of the RSA enVision system. The NIC_View alerts users to problems within the enVision software environment.

node

An appliance in an RSA enVision site.

output action

Configured notification method for alerts. The primary output actions are SMTP, SNMP, SNPP, Instant Messenger, syslog, run a command, text file, and task triage.

Overview module

The RSA enVision module that provides tools to configure the enVision platform and monitor system health and performance.

RC

See Remote Collector.

Remote Collector (RC)

An optional component of an RSA enVision multiple appliance site that captures incoming events at a remote location. A Remote Collector (RC) runs on its own appliance. Up to 16 RCs can be associated with a site.

Reports module

The RSA enVision module that provides tools to run standard network security and traffic analysis reports, or create and run custom reports.

single appliance site

An RSA enVision site in which all enVision components (Application, Collector, and Database) are on one appliance.

site

The basis on which the RSA enVision platform is deployed. Each site consists of three main components: Application Server, Collector, and Database Server.

site name

The name of the site, defined during the configuration of the RSA enVision platform.

standard report

Reports that are supplied within the RSA enVision platform for compliance, correlated alerts, event sources, as well as for task triage, and vulnerability and asset management.

task escalation

A function that allows users to send tasks to an external application, such as a ticketing system, for offline investigation.

task triage

A feature that allows users to group events into tasks for the purpose of investigation. Tasks can be further analyzed in the RSA enVision Event Explorer module, escalated to an external ticketing system, or both.

trace view

A set of parameters that define the information that is displayed in the form of tables and charts. The two forms of trace views are standard and advanced trace views.

UDC

See Universal Device Collection.

Universal Device Collection (UDC)

Allows the RSA enVision platform to collect log data from any event source that logs through SNMP, ODBC, or File Reader.

VAM

See vulnerability and asset management.

VDB

See Vulnerability Knowledge Database.

view

An administrator-defined set of event sources, messages, correlation rules, and criteria, within a single site, for which the RSA enVision platform issues alerts.

vulnerability and asset management

A feature that provides unified management of assets and vulnerability incident analysis.

Vulnerability Knowledge Database (VDB)

An embedded repository of vulnerability information derived from the National Vulnerability Database (NVD).

watchlist

A named collection of strings that represent a list of like-values. A watchlist can easily function as a filter for events in reporting and alerting.

Index