

CA Spectrum[®] Infrastructure Manager

Event Configuration User Guide

r9.2



This documentation and any related computer software help programs (hereinafter referred to as the "Documentation") are for your informational purposes only and are subject to change or withdrawal by CA at any time.

This Documentation may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA. This Documentation is confidential and proprietary information of CA and may not be used or disclosed by you except as may be permitted in a separate confidentiality agreement between you and CA.

Notwithstanding the foregoing, if you are a licensed user of the software product(s) addressed in the Documentation, you may print a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all CA copyright notices and legends are affixed to each reproduced copy.

The right to print copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO THE END USER OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and is not modified in any way by the terms of this notice.

The manufacturer of this Documentation is CA.

Provided with "Restricted Rights." Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Copyright © 2010 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

CA Product References

This document references the CA Spectrum® Infrastructure Manager (CA Spectrum).

Contact CA

Contact Technical Support

For your convenience, CA provides one site where you can access the information you need for your Home Office, Small Business, and Enterprise CA products. At <http://ca.com/support>, you can access the following:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Provide Feedback

If you have comments or questions about CA product documentation, you can send a message to techpubs@ca.com.

If you would like to provide feedback about CA product documentation, complete our short [customer survey](#), which is also available on the CA Support website, found at <http://ca.com/docs>.

Contents

Chapter 1: CA Spectrum Event and Alarm Concepts	9
About Alarms and Events	9
Alerts	9
Events	10
Event Codes	10
Alarms	11
Chapter 2: Getting Started with Event Configuration	13
Before You Begin	13
Obtaining a Developer ID	13
Mapping Traps to CA Spectrum Events	14
Preserving Customizations Across Upgrades of CA Spectrum	14
Starting the Event Configuration Application	15
Overview of the User Interface	17
Loading All Events from All Landscapes	18
Saving Events to Landscapes	18
Synchronizing Events in a Distributed Environment	20
Synchronizing Event Disposition Files on SpectroSERVERs	20
Synchronizing Event and Alarm Support Files on OneClick Web Servers	23
Updating the Overall Alert and Event System for a Landscape	24
Add and Remove Columns from the Events Table	25
Logging Event-Related Errors	25
Chapter 3: Working with Events and Alarms	29
Finding Events	29
Create Events from Scratch	30
Create Events from a Copy	31
About Configuring Events	32
Entering an Event Message	33
Specify Event Options	40
Configure Events to Generate Alarms	42
Configure Events to Clear Alarms	57
Modify Events	59
Delete Custom Events	60

Chapter 4: Working with Event Rules **61**

Event Rules	61
Event Condition Rules	61
Event Pair Rules	62
Event Rate Rules	62
Event Series Rules	64
Event Counter Rules	64
Heartbeat Rules	64
Single Event Rules	64
Solo Event Rules	65
User-Defined Event Rules	65
Create Event Rules	65
Configuring Event Condition Rule Settings	66
Configuring Event Pair Rule Settings	70
Configuring Event Rate Rule Settings	73
Configuring Event Series Rule Settings	76
Configuring Event Counter Rule Settings	80
Configuring Heartbeat Rule Settings	83
Configuring Single Event Rule Settings	86
Configuring Solo Event Rule Settings	89
Copy Variable Values from Contributing Events to the Rule Output Event	91
Modifying Event Rules	92
Delete Event Rules	93

Chapter 5: Using Procedures in Event Processing **95**

Using Procedures in Event Disposition Files	95
Input Parameters	96
Return Values	99
Examples of Procedures in Event Disposition Files	99
Logging Errors in Event Procedures	101
Troubleshooting Event Procedures	102

Appendix A: AlertMap Files **103**

SNMP Trap Overview	103
About Mapping a Trap to a CA Spectrum Event	105
About Processing Alerts with AlertMap Files	106
AlertMap File Location	106
AlertMap File Syntax	107
Error Messages	110
SNMPv2 Support	111
InformRequest Support	111

How an SNMPv2 Trap is Mapped to a CA Spectrum Event	111
---	-----

Appendix B: Event Disposition Files **115**

About Event Disposition Files	115
Location of Event Disposition Files	116
File Syntax of Event Disposition Files	118
Generating Alarms	119
Generating Alarms for Events Based on the Values of Event Variables	121
Generating Alarms Unconditionally for Each Event	121
Generating Alarms That Users Cannot Clear	122
Generating Alarms That Are Not Persistent	122
Combining the U, N, and T Flags	122
Specify an Event Frequency	123
Specify an Event Duration	123
Clearing Alarms	124
Clear Alarms Created Without Event Discriminators	124
Clear Alarms Based on Event Discriminator Values	125
Examples of Event Maps That Clear Alarms	127
Clearing Alarms Regardless of Event Discriminator Values	128
About Defining Event Rules	129
Event Rule Syntax	129
EventPair Rule	130
EventPairTimeAttr Rule	131
EventRateWindow Rule	131
EventRateWindowAttrParams Rule	132
EventRateCounter Rule	132
EventSequence Rule	133
EventCombo Rule	134
EventComboInclusive Rule	135
EventCondition Rule	135
EventCounter Rule	147
Heartbeat Rule	147
SoloEvent Rule	149
SingleEvent Rule	150
Using Multiple Event Rules in a Single EventDisp Entry	150
Copy Event Variables from One Event to Another	151
Syntax Errors in EventDisp Files	157
Add Comments in EventDisp Files	158

Appendix C: Event Format Files **159**

About Event Format Files	159
--------------------------------	-----

Location of Event Format Files.....	159
Contents of an Event Format File	160
Appendix D: Event Table Files	161
About Event Table Files	161
Location of Event Table Files	161
Contents of an Event Table File	162
Appendix E: Probable Cause Files	163
About Probable Cause Files	163
Location of Probable Cause Files	163
Contents of a Probable Cause File	164
Glossary	165
Index	167

Chapter 1: CA Spectrum Event and Alarm Concepts

This section contains the following topics:

[About Alarms and Events](#) (see page 9)

[Alerts](#) (see page 9)

[Events](#) (see page 10)

[Alarms](#) (see page 11)

About Alarms and Events

CA Spectrum is a services and infrastructure management system designed to notify you if a fault has occurred on any managed element within the network infrastructure. One way that CA Spectrum accomplishes this is by receiving alerts from problem areas within the infrastructure and converting those alerts into events and alarms that are displayed in OneClick event and alarm views. More simply, CA Spectrum notifies you about significant occurrences on your network through the use of alerts, events, and alarms.

Alerts

An *alert* is an unsolicited message sent out by a managed element on a network. A more specific definition of an alert depends on the management protocol that is used to report the alert. In general, CA Spectrum uses SNMP as the management protocol to communicate with devices on a network. Alerts that are generated by an SNMP-compliant device are called *traps*.

You can configure managed elements that have enabled SNMP traps to direct their traps to the host machine that is running CA Spectrum, which automatically listens for SNMP traps. When a trap is received, CA Spectrum uses the trap's source IP address to identify the model in CA Spectrum's database associated with the managed element. Next, CA Spectrum maps the trap to a CA Spectrum event that is then generated and processed.

CA Spectrum has special handling for traps that are not mapped to specific CA Spectrum events, and for traps that occur on managed elements that are not modeled at the time the trap is received.

You map the traps sent by a managed element to specific CA Spectrum events using the MIB Tools application in OneClick. In fact, you must do this before you can create and modify the events and associated alarms using Event Configuration.

Note: For more information about how traps are mapped to events, see the *Modeling and Managing Your IT Infrastructure Administrator Guide*. For information on using MIB Tools, see the *Certification User Guide*.

Events

An *event* is a CA Spectrum object that indicates that something significant has occurred within CA Spectrum itself or within the managed environment. Events always occur in relation to a model. When CA Spectrum receives an alert from a managed element on the network, in response, it generates a CA Spectrum event for the corresponding model if the received trap is mapped to an event.

CA Spectrum also generates events automatically in various situations, for example, when models are created or destroyed, when CA Spectrum connects to or disconnects from a device application, and when contact with a managed element is established or lost.

CA Spectrum processes an event instance based on how the underlying event is configured. For example, the event instance might be logged by the Archive Manager in the Distributed Data Manager (DDM) database. It might also clear an alarm or generate another event using an event rule.

Network operators can view the list of current events in a landscape on the Events tab in OneClick. For a specific event, you can also view information such as a message describing the event and when the event was created.

As previously mentioned, you map the traps sent by a device to specific CA Spectrum events using the MIB Tools application in OneClick. However, you use Event Configuration to fully configure the event, that is, to define how the event should be processed, the event message to display to users in OneClick, and so on.

Event Codes

Every event has a unique event code. The event code is a 4-byte integer expressed in hexadecimal format. It has two parts:

- The first two bytes contain the developer ID of the developer who created the event
- The last two bytes identify the event with a unique number relative to all other event codes for that developer

CA Spectrum assigns event codes to all events created using MIB Tools or Event Configuration, and the next available event code is always used as the default code. In Event Configuration, you have the option of overriding the default code and specifying a different one.

Note: The event code 0x10000 represents a null event. This event cannot be generated. However, the null event can be used in an event rule that requires an event code as a parameter.

Alarms

An *alarm* is a CA Spectrum object that indicates that a user-actionable, abnormal condition exists in a model. CA Spectrum generates an alarm when a CA Spectrum event—typically generated as a result of a received trap—specifies that one should be created. CA Spectrum can also generate an alarm based on the results of a watch set up, or as a result of CA Spectrum detecting an abnormal situation not based on an event (for example, lost communication between a model and the managed element that it represents).

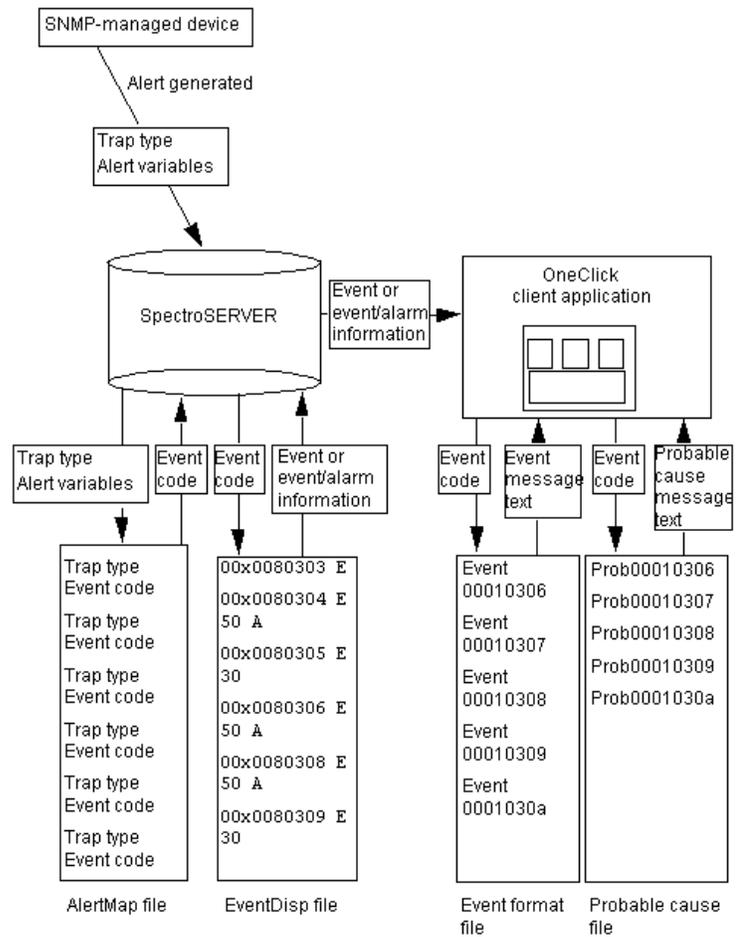
Network operators are alerted to alarms in various ways depending on how OneClick is configured. For example, the icon representing the model of the managed element (or a container model containing the managed element model) might change color. An audio message announcing the new alarm might also be played.

Operators can view the list of current alarms in a landscape on the Alarms tab in OneClick. For a specific alarm, you can also view detailed information, such as whether it has been acknowledged by a user, status, symptoms, probable causes, and the corrective actions that are recommended.

When the abnormal condition that caused the alarm ends, the corresponding alarm may be cleared automatically by another event or cleared manually by a user. Alarm notifications may be sent to other external third-party or internal CA Spectrum applications as appropriate.

For convenience, you can specify whether an event should generate an alarm (and the severity of the alarm) in MIB Tools when you map a trap to an event. However, you use Event Configuration to further configure the alarm (and to change the alarm severity if desired).

The following figure illustrates the flow of alerts, events, and alarms within CA Spectrum.



More information:

- [AlertMap Files](#) (see page 103)
- [Event Disposition Files](#) (see page 115)
- [Event Format Files](#) (see page 159)
- [Probable Cause Files](#) (see page 163)

Chapter 2: Getting Started with Event Configuration

This section contains the following topics:

- [Before You Begin](#) (see page 13)
- [Preserving Customizations Across Upgrades of CA Spectrum](#) (see page 14)
- [Starting the Event Configuration Application](#) (see page 15)
- [Overview of the User Interface](#) (see page 17)
- [Loading All Events from All Landscapes](#) (see page 18)
- [Saving Events to Landscapes](#) (see page 18)
- [Synchronizing Events in a Distributed Environment](#) (see page 20)
- [Updating the Overall Alert and Event System for a Landscape](#) (see page 24)
- [Add and Remove Columns from the Events Table](#) (see page 25)
- [Logging Event-Related Errors](#) (see page 25)

Before You Begin

Before you begin using Event Configuration to create events and alarms, do either or both of the following:

- [Obtain a developer ID](#) (see page 13) from CA.
- If you are adding trap support for a device that is not supported by default in CA Spectrum, [map the traps to new CA Spectrum events](#) (see page 14) using MIB Tools.

Obtaining a Developer ID

The first two bytes of any event code contains a developer ID. By default, this is the default developer ID provided with CA Spectrum. However, if you are creating events and alarms to support a new device management module or you are creating a Southbound Gateway integration, it is recommended that you obtain a unique, registered developer ID from CA so you can specify event codes for your events that begin with your unique developer ID. This lets you to easily recognize your custom code in OneClick and prevents potential conflicts with other CA Spectrum event codes.

To obtain a developer ID from CA, contact CA Spectrum Support toll-free at 877-428-6324. To be issued a developer ID, you must have purchased the Level 1 Toolkit.

Note: For information on the toolkit, see the *Integrator Guide*. To activate your developer ID, which is done using `SSdbload` with the `-d` option, see the discussion on loading developer information in the *Database Management Guide*.

Mapping Traps to CA Spectrum Events

If you are creating a new management module for a device that is not supported by default in CA Spectrum, it is recommended that you map the traps sent by the device to new CA Spectrum events (that are automatically created when the trap mappings are defined) using MIB Tools before you begin using Event Configuration. You can then launch Event Configuration directly from MIB Tools to fully configure the events and associated alarms.

If you create new events using Event Configuration instead of MIB Tools, you cannot subsequently use MIB Tools to map traps to them. Instead, you must manually specify the mappings in the `<$SPECROOT>/custom/Events/AlertMap` file on each SpectroSERVER in your environment. This is because the process of mapping traps to events using MIB Tools automatically creates new events with unique event codes. You cannot use MIB Tools to map traps to *existing* events that were previously created using Event Configuration.

AlertMap files are ASCII files that store the following:

- Mappings between the traps sent by a device and CA Spectrum events
- Mappings between the variable bindings that are sent with a trap and the event variables in the event that CA Spectrum generates on a model when the trap is received. Variable bindings can store attribute values in a MIB table, OIDs, or integer bit values.

Note: For more information about using MIB Tools to map traps to events, see the *Certification User Guide*.

Preserving Customizations Across Upgrades of CA Spectrum

There are several types of event and alarm configuration files that support event and alarm processing in CA Spectrum: alert mapping files, event disposition files, event format files, event table files, and probable cause files.

The files that are provided with CA Spectrum to support CA-authored events and alarms are installed in subfolders of the following folders:

<\$SPECROOT>/SS/CsVendor

<\$SPECROOT>/SG-Support

If you customize CA-authored events and alarms or create your own, and then you save the customizations to one or more landscapes, the event and alarm configuration files that define your customizations are installed in the following folder or in one of its subfolders:

<\$SPECROOT>/custom/Events

This means that the support files for your custom events and alarms are not overwritten or otherwise affected when you upgrade to a new version of CA Spectrum.

Starting the Event Configuration Application

You can start Event Configuration from the following locations:

- OneClick Console
- MIB Tools

Starting the Event Configuration Application from the OneClick Console

When you start Event Configuration from the OneClick Console, all of the events supported in the landscape (or landscapes if the environment is distributed) are loaded into the application.

To start Event Configuration from the OneClick Console

1. Launch the OneClick Console from the OneClick home page.
2. From the Tools menu, select Utilities, Event Configuration.

Starting the Event Configuration Application from MIB Tools

When you use MIB Tools to add trap support for a device that is not currently supported in CA Spectrum, you map the traps to new CA Spectrum events and specify whether those events should generate alarms. After you do this, typically you further customize the events and alarms, which you must do in Event Configuration. For this reason, you can start Event Configuration directly from MIB Tools.

When you start Event Configuration from MIB Tools, only the events that are associated with the traps you select are initially loaded into the application.

To start Event Configuration from MIB Tools

1. In the Navigation panel in MIB Tools, select the MIB that contains the traps that have been mapped to the events that you want to configure.

Note: For information on how to map traps to CA Spectrum events in MIB Tools, which you must do before you can configure the associated events in Event Configuration, see the discussion on trap support in the *Certification User Guide*.

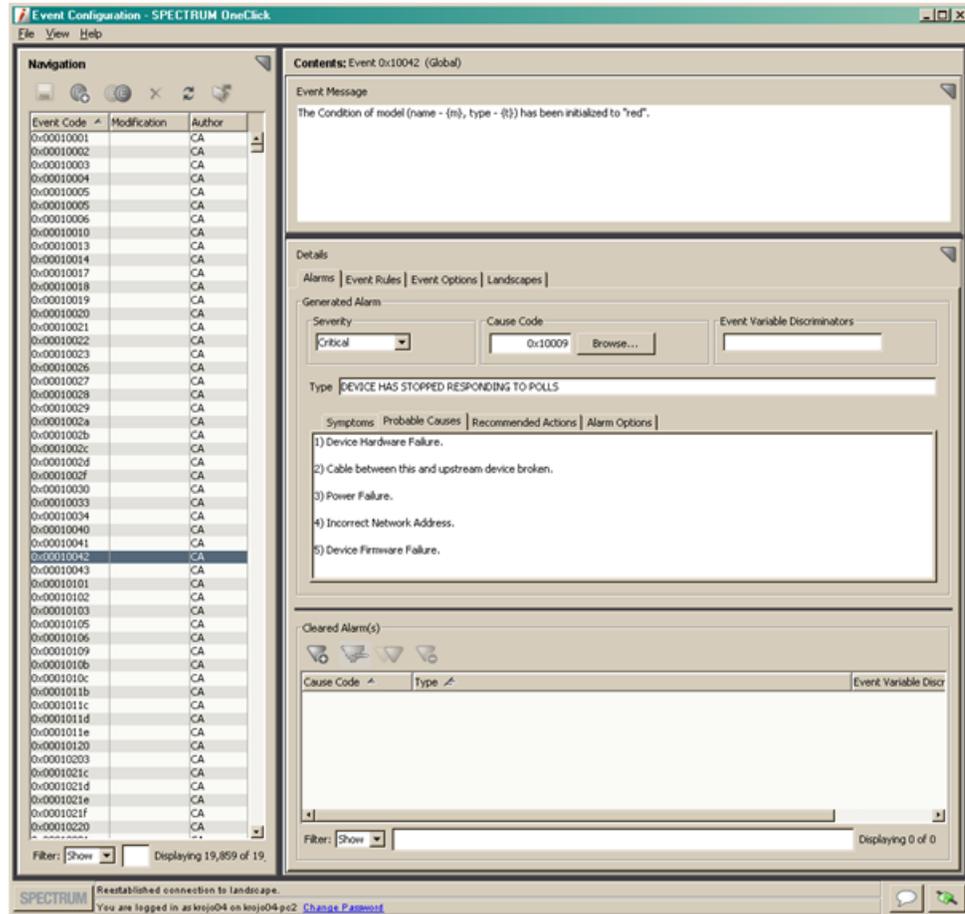
2. In the Contents panel, click the Map tab.
3. In the Trap Support table, select the mapped traps for which you want to configure events, and click  (Edit traps for selected items in the trap support table).

The Event Configuration application is started, and the events associated with the traps that you selected are listed by event code in the Navigation panel.

Note: At any time, you can load all of the events supported in the landscape (or landscapes if the environment is distributed) by clicking  (Reloads the list of events) on the Navigation panel.

Overview of the User Interface

You perform all event and alarm configuration tasks in the Event Configuration main window.



The window is divided into three panels:

- Navigation:** The Navigation panel lists all of the events currently loaded into Event Configuration.

The Modified column displays a checkmark beside any event that you have created or modified but not yet saved to a landscape. For more information, see [Saving Events to Landscapes](#) (see page 18).

- Contents:** The Contents panel displays a customizable event message that is displayed on the Events tab in OneClick for the event selected in the Navigation panel.

- **Details:** The Details panel provides access to the configuration information for the event selected in the Navigation panel. If the event should generate an alarm, you also configure the alarm in this panel.

Note: You can click the server connection icon () to access connection status information for all landscapes in the environment.

Loading All Events from All Landscapes

To load or reload all of the events supported in the landscape (or landscapes if the environment is distributed), click  (Reloads the list of events) on the Navigation panel.

This functionality is especially helpful if you started Event Configuration from MIB Tools to work with a limited set of events and you now require access to all events supported in all landscapes.

Saving Events to Landscapes

When you save new or modified events (and their disposed actions, such as alarms and event rules) to a landscape, Event Configuration updates the event disposition (EventDisp) file that defines them on the SpectroSERVER, and the SpectroSERVER flushes all existing event and alarm instances and reloads them using the most recent configuration information.

You can save all events or just selected ones. You can also save the changes to the local landscape only or, in a distributed environment, to some or all of the landscapes.

If you do not save your changes to one or more landscapes before exiting Event Configuration, the changes are discarded when you exit the application.

Important! The save process flushes and reloads all event rules including those that are in the middle of processing. As a consequence, the processing of events by active event rules is aborted, and all associated data (for example, counts for occurrences of contributing events) is lost.

To save events to the landscape in a single SpectroSERVER environment

1. Do one of the following:
 - If you want to save all modified or created events (and associated alarms), click Save All on the File menu.
 - If you want to save only specific events (and associated alarms), select the events in the table in the Navigation panel, and then click Save Selected on the File menu.
2. Click Yes.

To save events to one or more landscapes in a Distributed SpectroSERVER (DSS) environment

1. Do one of the following:
 - If you want to save all modified or created events (and associated alarms), click Save All on the File menu.
 - If you want to save only specific events (and associated alarms), select the events in the table in the Navigation panel, and then click Save Selected on the File menu.

If one or more SpectroSERVERs are unavailable, you are notified at this point with a warning, so you can cancel the process if desired.

Note: If there are unavailable SpectroSERVERs that will not receive the changes, you can synchronize the events and alarms on all landscapes later.

2. Click Yes to save the events and alarms to available SpectroSERVERs. Alternatively, click No to cancel the process.

The Select Landscapes dialog appears. By default, all available SpectroSERVERs are listed in the left list box, which means that all available SpectroSERVERs will receive the updated events and alarms.

3. If there are available landscapes to which you do not want to save the changes, select them in the left list box, and click the right-arrow button to move them to the right list box.
4. Click OK.

More information:

[Synchronizing Events in a Distributed Environment](#) (see page 20)

Synchronizing Events in a Distributed Environment

If you are running CA Spectrum in a distributed environment, typically you will need to perform one or both of the following tasks after making changes to events and alarms using Event Configuration and saving them to landscapes:

- Update the SpectroSERVERs in the environment that do not have the most current events and alarms. See [Synchronizing Event Disposition Files on SpectroSERVERs](#) (see page 20).
- Copy the event and alarm support files to all of the OneClick web servers in the environment. See [Synchronizing Event and Alarm Support Files on OneClick Web Servers](#) (see page 23).

Synchronizing Event Disposition Files on SpectroSERVERs

If you save event and alarm changes to only some of the SpectroSERVERs in a Distributed SpectroSERVER (DSS) environment, for example, because a server in the environment is unavailable, the result is that conflicts exist across the landscapes with respect to event and alarm configurations. To resolve these conflicts, you need to save the changes to the unavailable servers when they become available again, which is a task you can accomplish using the synchronization features in Event Configuration.

To begin resolving event (and alarm) conflicts across landscapes, first add the Conflict column to the table of events in the Navigation panel.

Next, examine the Conflict column and note any check marks (). If an event is configured differently across two or more landscapes, that is, if different event maps for the event are found in the event disposition (EventDisp) files on all of the SpectroSERVERs, the event is loaded into Event Configuration once for each unique event map in the DSS environment. In addition, to notify you of the disparity, a check mark is placed in the Conflict column beside each event for which different configurations have been retrieved.

As an example, note the two instances of event 0xf40004 in the following image.

The screenshot shows a software interface with a 'Navigation' pane on the left and a 'Contents' pane on the right. The 'Navigation' pane contains a table of events:

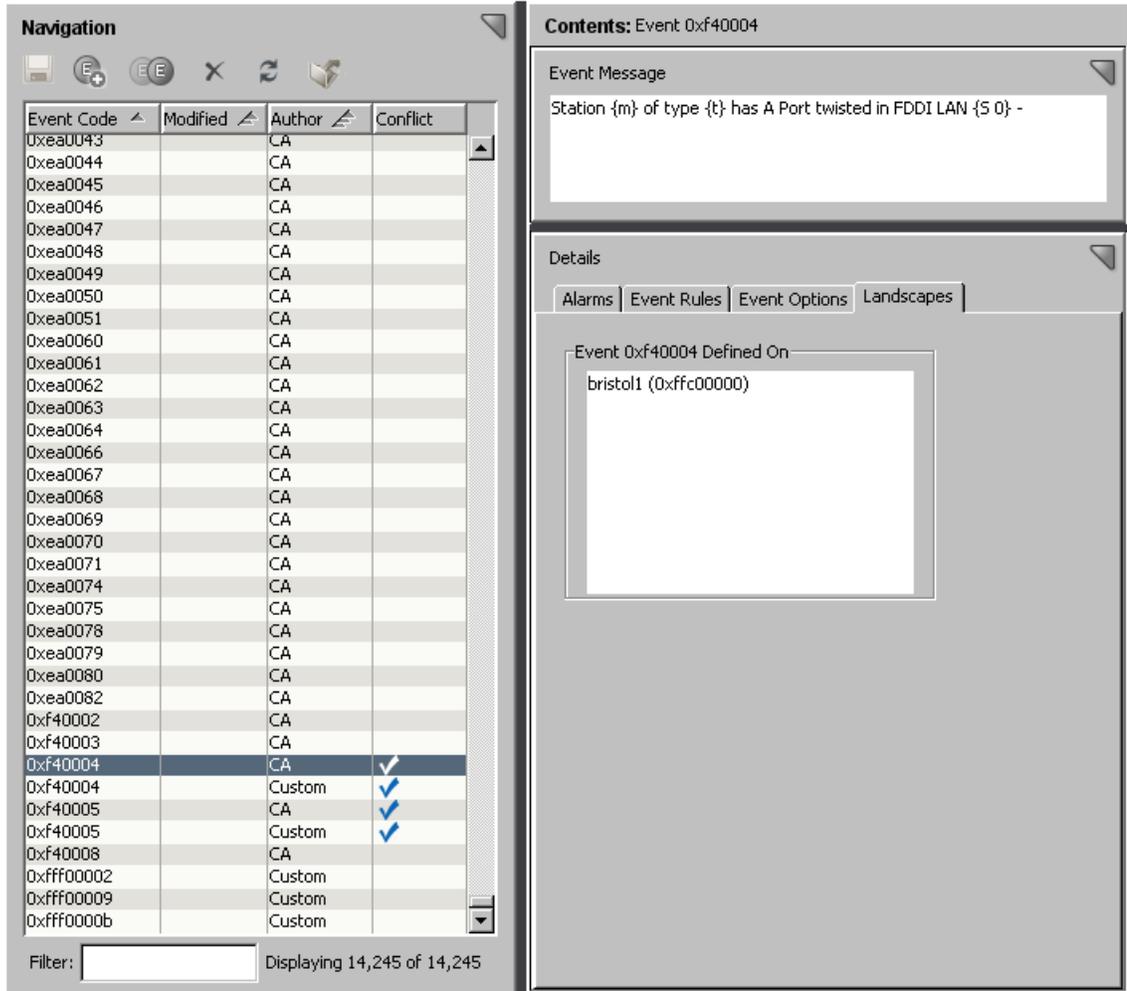
Event Code	Modified	Author	Conflict
0xea0043		CA	
0xea0044		CA	
0xea0045		CA	
0xea0046		CA	
0xea0047		CA	
0xea0048		CA	
0xea0049		CA	
0xea0050		CA	
0xea0051		CA	
0xea0060		CA	
0xea0061		CA	
0xea0062		CA	
0xea0063		CA	
0xea0064		CA	
0xea0066		CA	
0xea0067		CA	
0xea0068		CA	
0xea0069		CA	
0xea0070		CA	
0xea0071		CA	
0xea0074		CA	
0xea0075		CA	
0xea0078		CA	
0xea0079		CA	
0xea0080		CA	
0xea0082		CA	
0xf40002		CA	
0xf40003		CA	
0xf40004		CA	✓
0xf40004		Custom	✓
0xf40005		CA	
0xf40005		Custom	✓
0xf40008		CA	
0xfff00002		Custom	
0xfff00009		Custom	
0xfff0000b		Custom	

The 'Contents' pane shows details for 'Event 0xf40004'. The 'Event Message' is: 'Station {m} of type {t} has A Port twisted in FDDI LAN {S 0} -'. The 'Details' pane has tabs for 'Alarms', 'Event Rules', 'Event Options', and 'Landscapes'. The 'Landscapes' tab is selected, showing a list of landscapes where the event is defined: 'audi (0x4400000)', 'clubhouse (0x7a00000)', 'darlington (0x3e00000)', 'drillpress (0x2f00000)', 'formula3 (0xb400000)', 'hazard (0xc200000)', 'ihawk (0x5300000)', and 'primus (0x4500000)'.

The scenario shown in the preceding image is the result of customizing a CA-authored event (event 0xf40004) and then saving the customization to some but not all landscapes.

By selecting custom event 0xf40004 and examining the Landscapes tab (displayed automatically in a DSS environment) in the Details panel, you can identify the landscapes to which the custom event has been saved.

Similarly, by selecting CA-authored event 0xf40004 and examining the Landscapes tab, you can identify the landscapes to which the CA event has been saved. As another example, contrast the preceding image that shows the Landscapes tab for the custom event with the following image that shows the same tab for the CA-authored event.



By comparing the following:

- The configurations of the conflicting events
- The landscapes to which the conflicting events have been saved

You identify the events you need to save and the landscapes to which you need to save them to resolve a conflict. Once you have identified these, you can synchronize the landscapes.

The process of synchronizing the landscapes updates the event disposition (EventDisp) files on one or more SpectroSERVERs so that they match the event disposition file on the main location server in the DSS environment. If the main location server does not have the event disposition file that you want to use to update the rest of the SpectroSERVERs, you can use the Save commands instead, as described in [Saving Events to Landscapes](#) (see page 18).

Note: You manually designate a main location server when you install CA Spectrum. For more information about location servers, see the *Distributed SpectroSERVER Administrator Guide*.

To synchronize event disposition files across landscapes

1. Select Synchronize on the File menu.

The landscapes with event disposition files that differ from the file on the main location server are listed.

2. Select the landscapes to synchronize with the main location server, and click OK.

Note: You should also synchronize the event and alarm support files that reside on the OneClick web servers in the environment.

Note: For more information about synchronizing the event and alarm support files between fault tolerant servers, see the *Distributed SpectroSERVER Administrator Guide*.

More information:

[Synchronizing Event and Alarm Support Files on OneClick Web Servers](#) (see page 23)

[Add and Remove Columns from the Events Table](#) (see page 25)

Synchronizing Event and Alarm Support Files on OneClick Web Servers

When you create and configure events and alarms using Event Configuration, and then you save them to one or more landscapes, the following types of support files are created and updated automatically on only the OneClick web server to which you are currently connected:

Event format files

Event format files store the event messages that are displayed in OneClick for users. There is an event format file for every event that can be created by CA Spectrum.

Probable cause files

Probable cause files store the alarm messages that are displayed in OneClick for users. There is a probable cause file for every alarm that can be created by CA Spectrum and that appears on the Alarms tab in OneClick.

As a result, if you are running multiple OneClick web servers, you need to copy the following folders (which contain the event format files and the probable cause files) to the other OneClick servers in your distributed environment:

<\$SPECROOT>/custom/Events/CsEvFormat

<\$SPECROOT>/custom/Events/CsPCause

In addition, if you use the command line interface (CLI) commands `showalarms` or `showevents`, or you use CA Spectrum Alarm Notification Manager (SANM), then you will want to copy the contents of these same directories to <\$SPECROOT>/SG-Support on all of the SpectroSERVERs in your environment.

More information:

[Location of Event Format Files](#) (see page 159)

[Contents of an Event Format File](#) (see page 160)

[Location of Probable Cause Files](#) (see page 163)

[Contents of a Probable Cause File](#) (see page 164)

Updating the Overall Alert and Event System for a Landscape

As described in [Saving Events to Landscapes](#) (see page 18), you can update just the events (and their disposed actions, such as alarms or event rules) on one or more landscapes using the Save commands available on the File menu.

However, to update the overall alert and event system more broadly for a given landscape, in OneClick, click the Update Event Configuration button that is available on the SpectroSERVER Control subview of the Information tab on the VNM model. This action reloads the following:

- The alert maps defined in all custom and CA-authored AlertMap files.
- The event maps (including event rules) defined in all custom and CA-authored event disposition files.
- The event procedures defined in all custom and CA-authored event procedure definition files.
- The severity maps defined in all custom and CA-authored severity mapping files (which are used for alarms that are assigned an alarm severity level of Conditional).

- The event-related resource settings defined in the .vnmrc file for the SpectroSERVER.

Note: For information on these settings, see [Logging Event-Related Errors](#) (see page 25). For more information about the .vnmrc resource file, see the *Distributed SpectroSERVER Administrator Guide*.

- The parse maps defined in all custom and CA-authored parse map files.

Note: For information on parse map files, see the *Host System Resources Management User Guide*.

Important! This update process flushes and reloads all event rules including those that are in the middle of processing. As a consequence, the processing of events by active event rules is aborted, and all associated data (for example, counts for occurrences of contributing events) is lost.

Add and Remove Columns from the Events Table

You can modify the event information that is displayed in the table of events in the Navigation panel by adding or removing columns from the table.

For example, if you are working in a Distributed SpectroSERVER (DSS) environment, it can be helpful to add the Conflict column to the table so you can easily identify if there are existing events that are configured differently on different landscapes.

Note: To filter the events in the table based on a specific event property, the corresponding event property column must be displayed. The filtering mechanism checks the text string you specify against only the text in the *displayed* columns.

To add or remove columns from the events table

1. Right-click any column heading.

The Table Preferences dialog opens.

2. Click the Columns tab, and select the columns you want to display.

Note: If desired, you can also change the table sorting and font using the controls on the Sort and Font tabs.

3. Click OK.

Logging Event-Related Errors

When you create and configure events and alarms using Event Configuration and then save them to a landscape, the event and alarm processing instructions are written to configuration files referred to as event disposition files.

To help you resolve errors in event disposition files, which result in errors in event processing, you can write errors of different types to log files. You specify which types of errors to log, as well as the log files to which to write them, using several parameters in the VNM resource file (.vnmrc file) for the SpectroSERVER. See the subsections that follow for details on each parameter.

Several types of errors, such as syntax errors, are typically the result of *manual* modifications to event disposition files. To minimize these types of errors, it is recommended that you use Event Configuration, not a manual process, to create and configure events and alarms.

Important! If the SpectroSERVER encounters an error in an event map while parsing an event disposition file, that event map is ignored and, therefore, cannot be used to process the associated event.

Note: If you make changes to the event-related .vnmrc parameters described in this section, you must reload the parameters on the SpectroSERVER using the Update Event Configuration command on the Information tab of the VNM model for the changes to take effect. For more information, see [Updating the Overall Alert and Event System for a Landscape](#) (see page 24). Alternatively, you can restart the SpectroSERVER; this reloads all of the parameters in the .vnmrc file, not just the event-related ones. For more information on the .vnmrc file, see the *Distributed SpectroSERVER Administrator Guide*.

event_disp_error_file

If you set this parameter to the name of a text file, CA Spectrum writes any syntax errors or other errors that it encounters while parsing an event disposition file to that text file. Use the following syntax:

```
event_disp_error_file=<file name>
```

The text file is created in the <\$SPECROOT>/SS folder.

Alternatively, you can set the value as follows:

```
event_disp_error_file=stderr
```

This sends the output to the console window in the CA Spectrum Control Panel and to the <\$SPECROOT>/SS/VNM.OUT file.

Note: For more information on syntax errors, see [Syntax Errors in EventDisp Files](#) (see page 157).

event_custom_override_warnings

If you set this parameter to TRUE, when CA Spectrum encounters an event map for an event in a custom event disposition file, and an event map for the same event also exists in an event disposition file provided with CA Spectrum, it logs a warning to the text file specified in the event_disp_error_file parameter (if a file is specified). The default value is FALSE.

Setting this parameter to TRUE can be helpful if you need to determine which CA-authored events are being overridden by your custom events, for example, for troubleshooting purposes. Use the following syntax:

```
event_custom_override_warnings=TRUE
```

Note: To set this parameter, you must manually add it to the .vnmrc file.

enable_event_variable_warnings

By default, CA Spectrum does not log any warnings that are encountered while copying event variable values from one event to another during event rule processing. You can override this default behavior by setting the value of this parameter to TRUE. Use the following syntax:

```
enable_event_variable_warnings=TRUE
```

Note: To set this parameter, you must manually add it to the .vnmrc file.

event_duplicate_action_warnings

By default, when CA Spectrum encounters *identical* (and, therefore, duplicate) event maps for an event within an event disposition file or across multiple event disposition files on a SpectroSERVER, it logs a warning to the text file specified in the event_disp_error_file parameter (if a file is specified).

Typically, this default behavior is desirable. However, you can override it by setting the value of this parameter to FALSE. Use the following syntax:

```
event_duplicate_action_warnings=FALSE
```

Note: To set this parameter, you must manually add it to the .vnmrc file.

event_disp_default_log

For events that you create and manage using Event Configuration, you specify whether they are logged in the Distributed Data Manager (DDM) database using the Store Event in Historical Database event option in Event Configuration. See [Specify Event Options](#) (see page 40) for more information.

By default, events that do not have event maps in event disposition files are logged in the DDM database. You can override this default behavior by setting the value of this parameter to FALSE. Use the following syntax:

```
event_disp_default_log=FALSE
```

Typically, most events have event maps in event disposition files. However, in rare situations, this is not the case. For example, if you mapped a trap to a CA Spectrum event *manually* instead of using the recommended method of using MIB Tools, and then you inadvertently neglected to define an event map for the event in an event disposition file, the event is still logged in the DDM database depending on the value of this .vnmrc parameter.

procedure_error_file

If you set this parameter to the name of a text file, CA Spectrum writes any errors that it encounters while parsing a procedure file to that text file. Use the following syntax:

```
procedure_error_file=<file name>
```

The text file is created in the <\$SPECROOT>/SS folder.

Alternatively, you can set the value as follows:

```
procedure_error_file=stderr
```

This sends the output to the console window in the CA Spectrum Control Panel and to the <\$SPECROOT>/SS/VNM.OUT file.

If the event_disp_error_file parameter is set, the errors encountered in procedures that are executed from events are written to the file specified in event_disp_error_file instead of to the file specified in this parameter. Errors encountered in other types of procedures, for example, in those used for diagnostics, are still written to the file specified in this parameter.

Note: To set this parameter, you must manually add it to the .vnmrc file.

Chapter 3: Working with Events and Alarms

This section contains the following topics:

- [Finding Events](#) (see page 29)
- [Create Events from Scratch](#) (see page 30)
- [Create Events from a Copy](#) (see page 31)
- [About Configuring Events](#) (see page 32)
- [Modify Events](#) (see page 59)
- [Delete Custom Events](#) (see page 60)

Finding Events

To find an event, you can filter the list of events in the Navigation panel to include only those with displayed property values that include a specific text string. For example, *if the Type column is displayed*, you can enter “timeout” to filter the list to include only events that generate alarms that include the word (in uppercase or lowercase) in the alarm type text string (alarm title).

An event that is mapped to a Trap will have the Trap Event column checked.

To find events

1. If necessary, click  (Reloads the list of events) to update the event table in the Navigation panel to include all events in the distributed environment.

Note: Typically, you only need to reload the list of events if you started Event Configuration from MIB Tools. Starting Event Configuration in this manner only loads into Event Configuration the specific events you selected in MIB Tools.

2. Verify that the event properties that you want to search against are displayed in the events table. If this is not the case, add the appropriate table columns as described in [Adding and Removing Columns from the Events Table](#) (see page 25).

Note: Only visible table columns are included in the filtering process.

3. In the Filter field, enter the text string to search for in the event table.

The list of events in the table is filtered to include only those that have displayed property values that include the text string you specified.

Create Events from Scratch

You can create new events from scratch.

Note: If you are creating a new management module for a device that is not supported by default in CA Spectrum, it is recommended that you map the traps sent by the device to new CA Spectrum events (that are automatically created when the trap mappings are defined) using MIB Tools before you begin using Event Configuration. You can then launch Event Configuration directly from MIB Tools to fully configure the events and associated alarms. For more information, see [Mapping Traps to CA Spectrum Events](#) (see page 14).

To create a new event

1. In the Navigation panel, click  (Creates a new event).

The Create Event dialog appears.

2. Enter an event code or accept the default event code.

Note: The event code is a 4-byte integer expressed in hexadecimal format. The first 2 bytes contain the developer ID, and the last 2 bytes identify the event with a unique number. While the default code is unique, it is recommended that you enter a code that begins with your CA-assigned developer ID. This lets you to easily recognize your custom code in OneClick and prevents potential conflicts with other CA Spectrum event codes. The event code 0x10000 represents a null event. This event cannot be generated. However, the null event can be used in an event rule that requires an event code as a parameter.

3. Enter an event message as described in [Entering an Event Message](#) (see page 33). (You can also modify the message after the event is created.)

Keep in mind that most of the information that a OneClick user receives about an event is via the message text that is affiliated with that event. For this reason, provide as much information about the event as possible in the message.

4. (Optional) Enter the Vendor to specify the developer, vendor, or manufacturer.

Note: The Vendor is available in the directory under `<$SPECROOT>/custom/Events` that contains the [EventDisp](#) (see page 116) file. Event options (and other event processing information) are stored in event configuration files referred to as event disposition files.

5. Click OK.

The new event is added to the table of events in the Navigation panel. The event is displayed in bold. The Modification column displays *New*.

Note: The event is marked *New*. However, the event is not saved. You can make more updates such as create more events, modify existing events, or delete events, and save all the updates at one time.

6. Configure the event, as described in [Configuring Events](#) (see page 32).
7. (Optional) Add event rules to the event, as described in [Creating an Event Rule](#) (see page 65).
8. Save the changes to one or more landscapes, as described in [Saving Events to Landscapes](#) (see page 18).

The events appear in the normal font and are not marked in the Modification column.

Create Events from a Copy

You can create new events by copying existing events.

Note: If you are creating a new management module for a device that is not supported by default in CA Spectrum, it is recommended that you map the traps sent by the device to new CA Spectrum events (that are automatically created when the trap mappings are defined) using MIB Tools before you begin using Event Configuration. You can then launch Event Configuration directly from MIB Tools to fully configure the events and associated alarms. For more information, see [Mapping Traps to CA Spectrum Events](#) (see page 14).

To create an event from a copy

1. In the Navigation panel, select the event you want to copy, and click  (Copies the selected event).

The Copy Event dialog opens.

2. Enter an event code or accept the default event code.

Note: The event code is a 4-byte integer expressed in hexadecimal format. The first 2 bytes contain the developer ID, and the last 2 bytes identify the event with a unique number. While the default code is unique (even with respect to the event you are copying), it is recommended that you enter a code that begins with your CA-assigned developer ID. This lets you to easily recognize your custom code in OneClick and prevents potential conflicts with other CA Spectrum event codes. The event code 0x10000 represents a null event. This event cannot be generated. However, the null event can be used in an event rule that requires an event code as a parameter.

3. Revise the event message as appropriate for the new event, as described in [Entering an Event Message](#) (see page 33). (You can also modify the message after the event is created.)

Keep in mind that most of the information that a OneClick user receives about an event is via the message text that is affiliated with that event. For this reason, provide as much information about the event as possible in the message.

4. Click OK.

The new event is added to the table of events in the Navigation panel. The event displays in bold. The Modification column displays *New*.

Note: The event is marked *New*. However, the event is not saved. You can make more updates such as create more events, modify existing events, or delete events, and save all the updates at one time.

5. Configure the event, as described in [Configuring Events](#) (see page 32).
6. (Optional) Add event rules to the event, as described in [Creating an Event Rule](#) (see page 65).
7. Save the changes to one or more landscapes, as described in [Saving Events to Landscapes](#) (see page 18).

The events appear in the normal font and are not marked in the Modification column.

About Configuring Events

When you configure an event, you specify the following:

- The message that is displayed to users in OneClick when the event occurs. See [Entering an Event Message](#) (see page 33).
- The scope of the event (global or model type-specific) and whether the event is logged in the Distributed Data Manager (DDM) database by the Archive Manager for historical and reporting purposes. See [Specifying Event Options](#) (see page 40).
- Whether the event should generate an alarm. See [Configuring Events to Generate Alarms](#) (see page 42).
- Whether the event should clear one or more alarms. See [Configuring Events to Clear Alarms](#) (see page 57).

Note: You can also create event rules that are activated (triggered) by an event. For example, some types of events can be tolerated and do not indicate a problem unless the frequency at which they are generated reaches a specific threshold within a specific amount of time. You can create a rule that watches for this scenario, and when it occurs, generates another event (and associated alarm) in response.

More information:

[Entering an Event Message](#) (see page 33)

[Specify Event Options](#) (see page 40)

[Configure Events to Generate Alarms](#) (see page 42)

[Configure Events to Clear Alarms](#) (see page 57)

[Working with Event Rules](#) (see page 61)

Entering an Event Message

The *event message* is the message that is displayed on the Events tab in OneClick when the event occurs.

Component Detail: Cisco-FRX of type Cisco4500

Information | Host Configuration | Root Cause | Interfaces | Performance | Neighbors | Alarms | Events | Attributes

Filter: Displaying 2 of 2

2 event(s) from Feb 10, 2008 10:41:35 AM EST - now

Created On	Name	Event	Created By
Feb 10, 2008 12:11:40 PM EST	Cisco-FRX	Contact has been lost with model Cisco-FRX of type Rtr_Cisco.	System
Feb 10, 2008 12:11:19 PM EST	Cisco-FRX	Device Cisco-FRX of type Rtr_Cisco has stopped responding to polls and/or external requests. An alarm will be generated.	System

When you compose an event message, you can use plain text as well as variables that reference specific data about the generated event. For descriptions of each variable and the proper syntax to use when including them, see the subsections that follow.

Note: Event messages are stored in event configuration files referred to as event format files.

More information:

[About Event Format Files](#) (see page 159)

Variable Descriptions and Syntax

This section provides information on the event variables that you can use when you define event messages. When you include an event variable in a message, use the syntax defined here.

In the sections that follow, the # sign represents the event variable ID that is mapped to the OID of the variable binding sent with the trap. This assignment is made in the OID map in an AlertMap file that is automatically created by MIB Tools when you map a trap to an event.

You can construct any type of message using event variable IDs. The only requirement is that the variables that you use in the message are of the proper data type.

Note: 0x12a63 is a reserved event variable ID that is used for a web context URL.

{d "%w- %d %m-, %Y - %T"}

This is the variable for the date string. It should be included in every event message. By including this variable exactly as shown, you tell CA Spectrum to capture the time and date that the alert is received.

Note: You do not need to include this variable if you are entering the event message using Event Configuration, as the application automatically inserts the variable in the event format file that it creates when you save the event to a landscape.

Date/Time specifier options:

%u

Use GMT time not local time. Must appear first in format.

%%

Writes % to buffer.

%d

Writes day-of-month 01..31 to buffer.

%d-

Writes day-of-month 1..31 to buffer.

%D

Writes date dd/mm/yy to buffer.

%H

Writes hour-of-day 00..23 to buffer.

%h

Same as %m.

%j

Writes day-of-year 1..366 to buffer.

%m+

Writes full month name to buffer.

%m-

Writes abbreviated month name to buffer.

%m*

Writes month index 1..12 to buffer.

%m

Writes month index 01..12 to buffer.

%M

Writes minute 00..59 to buffer.

%S

Writes second 00..59 to buffer.

%T

Writes time hh:mm:ss to buffer.

%w+

Writes full weekday name to buffer.

%w-

Writes abbreviated weekday name to buffer.

%w

Writes weekday index 1..7 to buffer.

%y

Writes year 00..99 to buffer.

%Y

Writes year 0000..9999 to buffer.

{t}

This variable inserts the model type name in the message. The (t) variable is defined internally and is not configurable.

{m}

This variable inserts the model name in the message. The (m) model name variable is defined internally and is not configurable.

{e}

This variable inserts the event code in the message. The (e) variable is defined internally and is not configurable.

{u}

This variable inserts the user name in the message. The (u) variable is defined internally and is not configurable.

{T <event table file name> #}

Inserts a text string that is associated with a MIB table attribute value. The association between the attribute value and the text string is defined in an event table file that is automatically created by MIB Tools when you map the trap to the event. For more information, see [Referencing Attribute Values in a MIB Table](#) (see page 37).

{Y <event table file name> #}

Inserts a text string that is associated with an OID. The association between the attribute value and the text string is defined in an event table file that is automatically created by MIB Tools when you map the trap to the event. For more information, see [Referencing OIDs](#) (see page 38).

{Z <event table file name> #}

Inserts a text value associated with an integer bit value. The association between the integer bit value and the text string is defined in an event table file that is automatically created by MIB Tools when you map the trap to the event. For more information, see [Referencing Integer Bit Values](#) (see page 39).

{o #}

Inserts an object ID.

{O #}

Inserts an octet string.

{X #} or {x #}

Inserts an octet string displayed in hexadecimal format.

{S #}

Inserts a text string.

{B #}

Inserts a Boolean value. Zero denotes false. Any other value denotes true.

{I #}

Inserts an integer.

{L #}

Inserts a Counter64 counter.

{U #}

Inserts an unsigned integer or Counter64 counter.

{R #}

Inserts a real number in the range: 10E37 to 10E37.

{H #}

Inserts a 32 bit hex number with a 0x prefix.

{K #}

Converts a DateAndTime attribute value from an octet string to a text string, and inserts the formatted text string.

{G #}

Calculates and inserts the device up time based on the value of the event variable (#). The value is displayed as days+hours:mins:secs.

{D #}

Used with an event variable (#), which contains an integer representing the number of seconds since 1969. Converts that value to a string that represents the date and time.

Referencing Attribute Values in a MIB Table

If a variable binding that is sent with a trap contains an attribute value from a MIB table, you can use it in an event message. To do so, you must use the proper syntax and reference the following:

- The event variable to which the OID of the variable binding is mapped.
- The event table file that contains the enumerated attribute values and the associated text strings to use in event messages. (Event table files are automatically created by MIB Tools when you map traps to new CA Spectrum events.)

As an example, assume you have an event table file named BeaconType that associates the following attribute values with corresponding text strings:

```
0x00000001 Reconfiguration
0x00000002 Signal-Loss
0x00000003 Bit-Streaming
0x00000004 Contention-Streaming
0x000000ff None
```

To reference these values in an event message, use the following syntax:

```
{T BeaconType 2}
```

T

Indicates that you are inserting a MIB table attribute whose values are enumerated in an event table file.

BeaconType

Specifies the name of the event table file that contains the enumerated values.

2

Specifies the event variable number that is mapped (in the AlertMap file) to the OID of the variable binding sent with the trap. CA Spectrum takes the value of the event variable and retrieves the corresponding text string defined in the event table file. For example, if the event variable above, 2, stored the value 3, the text "Bit-Streaming" would be rendered in the event message.

More information:

[About Event Table Files](#) (see page 161)

Referencing OIDs

If a variable binding that is sent with a trap contains an OID, you can use it in an event message. To do so, you must use the proper syntax and reference the following:

- The event variable to which the OID of the variable binding is mapped.
- The event table file that contains the enumerated OID values and the associated text strings to use in event messages. (Event table files are automatically created by MIB Tools when you map traps to new CA Spectrum events.)

As an example, assume you have an event table file named NewTable that associates the following attribute values with corresponding text strings:

```
1.3.6.1.4.1.1563.1.2.1.1.3.2.36.2.6 dot6  
1.3.6.1.4.1.1563.1.2.1.1.3.2.36.2.5 dot15  
1.3.6.1.4.1.1563.1.2.1.1.3.2 dot7
```

To reference these values in an event message, use the following syntax:

```
{Y NewTable 2}
```

Y

Indicates that you are inserting the value of a variable binding whose possible OID values are enumerated in an event table file.

NewTable

Specifies the name of the event table file that contains the enumerated values.

2

Specifies the event variable number that is mapped (in the AlertMap file) to the OID of the variable binding sent with the trap. CA Spectrum takes the value of the event variable and retrieves the corresponding text string defined in the Event Table file. For example, if the event variable above, 2, stored the value 1.3.6.1.4.1.1563.1.2.1.1.3.2, the text "dot7" would be rendered in the event message.

More information:

[About Event Table Files](#) (see page 161)

Referencing Integer Bit Values

If a variable binding that is sent with a trap contains an integer bit value, you can use it in an event message. To do so, you must use the proper syntax and reference the following:

- The event variable to which the OID of the variable binding is mapped.
- The event table file that contains the enumerated integer bit values and the associated text strings to use in event messages.

Note: Event table files are automatically created by MIB Tools when you map traps to new CA Spectrum events.

As an example, assume you have an event table file named NewBitTable that associates the following integer bit values with corresponding text strings:

```
1 dsx1NoAlarm
2 dsx1RcvFarEndLOF
3 dsx1XmtFarEndLOF
4 dsx1RcvAIS
```

To reference these values in an event message, use the following syntax:

```
{Z NewBitTable 2}
```

Z

Indicates that you are inserting the value of a variable binding whose possible integer bit values are enumerated in an event table file.

NewBitTable

Specifies the name of the event table file that contains the enumerated values.

2

Specifies the event variable number that is mapped (in the AlertMap file) to the OID of the variable binding sent with the trap. CA Spectrum takes the value of the event variable and retrieves the corresponding text string defined in the Event Table file. For example, if the event variable above, 2, stored the value 4, the text "dsx1RcvAIS" would be rendered in the event message.

More information:

[Event Table Files](#) (see page 161)

Example Event Message

The following is a sample event message:

```
{d "%w- %d %m-, %Y - %T"} A device {m} of type {t} has reported a Firewall trap has occurred. {S 1} contains the name of the last trap sent via fw. - (event [{e}])
```

When the message is displayed on the Events tab in OneClick, it is rendered as follows:

- {d "%w- %d %m-, %Y - %T"} is replaced with the date and time
- {m} is replaced with the model name
- {S 1} is replaced with a string value (S for string data type) from a variable binding
- {t} is replaced with the model type
- {e} is replaced with the event code

Specify Event Options

You can specify the following options for an event:

- Whether the event is logged in the Distributed Data Manager (DDM) database by the Archive Manager for historical and reporting purposes.

Events for a model that are not logged in the DDM database are displayed on the Events tab in OneClick only if they are generated while the Events tab for that model is displayed.

- Whether the event is global or specific to one or more model types.

Global events are those that are generated for all models of all model types regardless of the developer who created the model type. Examples of global events include “link down” or “cold start” events.

If an event is *specific to a model type*, it is generated only for models of specific model types (for example, for a device model type that supports a proprietary MIB).

- The Vendor field appears only for those events that are defined for a vendor. It appears as read-only only for events that are defined under a vendor directory. It is configurable for a new event until the event has been saved, then it will appear as read-only.

Note: Event options (and other event processing information) are stored in event configuration files referred to as event disposition files.

To specify options for an event

1. Select the event in the Navigation panel.
2. In the Details panel, click the Event Options tab.
3. If you want the event to be logged in the Distributed Data Manager (DDM) database by the Archive Manager, select Store Event in Historical Database.
4. Under Scope, specify the scope of the event:
 - If the event is global, select Global.
 - If the event is specific to a model type, select Model Type. Then, in the Select Model Type dialog, select the name of one or more model types to which the event should apply, and click OK. (Use the CTRL key to select multiple model types.)

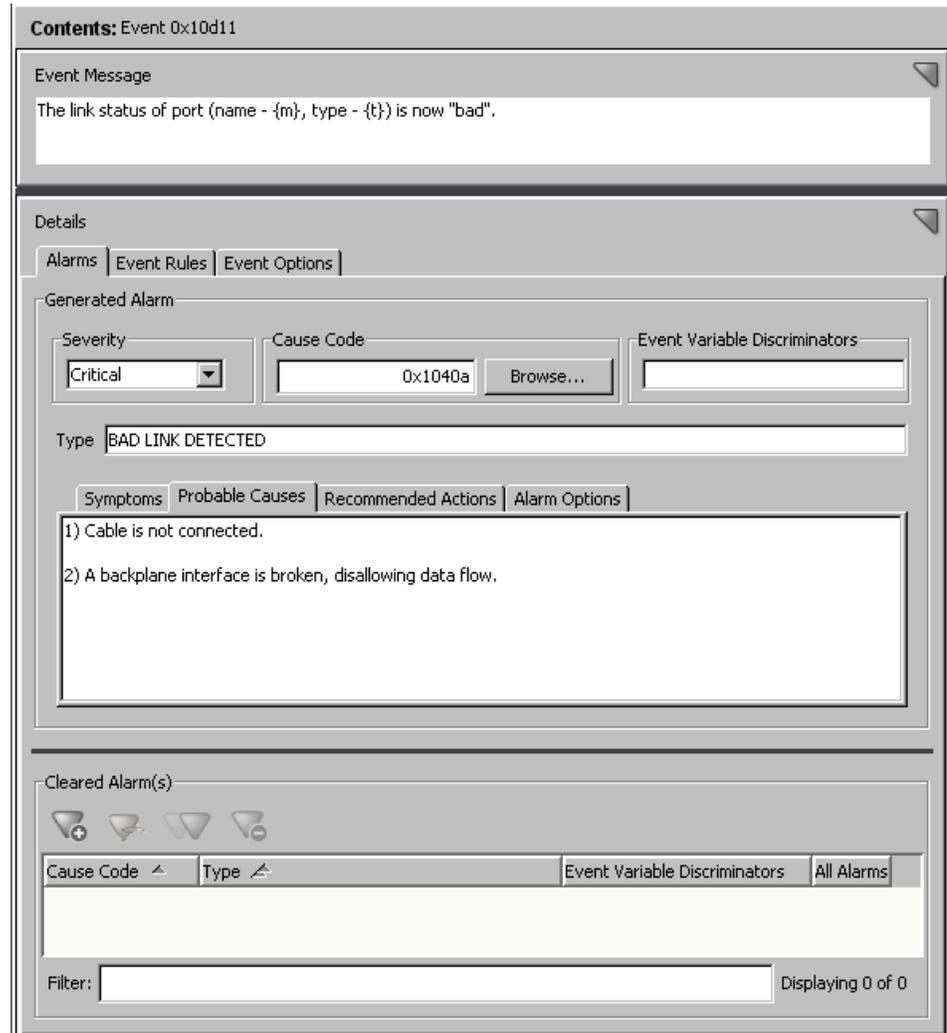
Note: Changing the scope of an event does not modify the event; instead, a duplicate event with the same event code but a different event scope is automatically created. As a result, when you save both events to a landscape, two event maps in two different event disposition files are created. This lets you to specify event processing for the event and apply those instructions globally, and then override those processing instructions, for the same event, for specific model types. When an event is processed by CA Spectrum, the event maps in model type-specific event disposition files take precedence over the event maps for the same events in global event disposition files. If you change the scope of an event, and you do not require the original event, you can delete it if it is a custom event.

More information:

[About Event Disposition Files](#) (see page 115)

Configure Events to Generate Alarms

You can specify that the currently selected event should generate an alarm, and you can configure the alarm itself using the Alarms tab in the Details panel, which is shown in the following image.



When you configure an event to generate an alarm and then save that change to a landscape, you create a mapping between the event and the alarm in a configuration file referred to as an event disposition file.

To configure an event to generate an alarm

1. Select the event in the Navigation panel, and then click the Alarms tab in the Details panel.

Note: Under Generated Alarm, the value for Severity is None. This indicates that the event does not generate an alarm.

2. Select an alarm severity other than None from the Severity drop-down list. For help with this step, including descriptions on the different levels of severity, see [Specifying an Alarm Severity](#) (see page 45).
3. (Optional) For Cause Code, change the alarm cause code (the 8-digit, hexadecimal code that identifies the cause of the alarm). For help with this step, see [Specifying an Alarm Cause Code](#) (see page 47).

As a convention, events that generate alarms typically use their event codes as alarm cause codes. For this reason, the event code of the event is the default alarm cause code of the alarm.

4. For Event Variable Discriminators, enter a comma-separated list of event variable IDs if you want to use the values of the variables in the event to determine whether to generate the alarm. You must enter each ID; ranges of IDs are not supported.

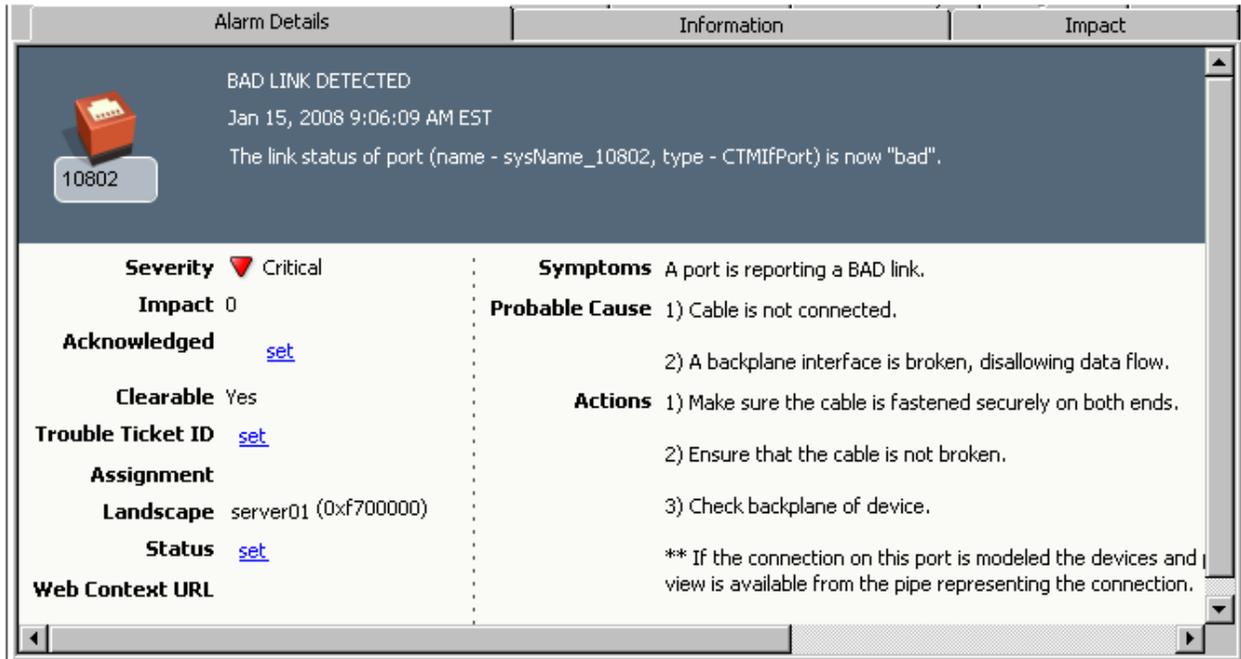
Note: By default, CA Spectrum does not generate a new alarm each time the same event occurs if an alarm already exists for that event on the model. You can use event discriminators or alarm options to change this default behavior.

For example, if the event generates alarm 0x3b10011, and you enter "1,3" for Event Variable Discriminators, then if an existing alarm 0x3b10011 already exists on the model, another alarm is not generated unless the values for *both* event variables 1 and 3 are *different* in the new event instance when compared to current alarms on the model generated from the same event.

For more help with this step, see [Using Event Variable Discriminators to Generate Alarms](#) (see page 50).

5. For Type, enter a text string that identifies the type of the alarm, for example, "BAD LINK DETECTED."

The text string you enter for Type is displayed as the alarm title in OneClick, as shown in the following image. For enhanced readability in OneClick, it is recommended that you enter the text string in capital letters.



6. Specify the symptoms, probable causes, and recommended corrective actions for the alarm, respectively, on the Symptoms, Probable Causes, and Recommended Actions tabs. This information is displayed on the Alarm Details tab in OneClick, as shown in the preceding image. For help with this step, see [Specifying Symptoms, Causes, and Recommended Actions](#) (see page 49).
7. Click the Alarm Options tab, and specify advanced options for the alarm. For help with this step, see [Specifying Alarm Options](#) (see page 49).
8. If the currently selected event should also *clear* one or more alarms, specify the alarms in the Cleared Alarm(s) area of the Contents panel. For help with this step, see [Configuring an Event to Clear Alarms](#) (see page 57).

More information:

[About Event Disposition Files](#) (see page 115)

Specify an Alarm Severity

The following lists the alarm severity levels used in CA Spectrum. Each severity level is associated with a color-coded condition that is displayed on the model. When an alarm of the specified severity is asserted on a model, the condition color is displayed on the model's icon to reflect the alarm status.

Normal (0)

Color-coded condition: Green

Indicates that contact has been made with the device, and the device is operating normally. There are no alarms associated with the device. If an event generates an alarm, but a severity for the alarm is not specified (for example, if you have created the supporting EventDisp configuration file manually and inadvertently omitted a severity), CA Spectrum assigns the alarm a severity of Normal.

Minor (1)

Color-coded condition: Yellow

Indicates that an abnormal situation exists, but no immediate action is required. This level of severity is also used for alarms created only to convey information, such as "Duplicate IP."

Major (2)

Color-coded condition: Orange

Indicates that a loss of service has occurred or is impending. Action is required within a short period of time.

Critical (3)

Color-coded condition: Red

Indicates that a loss of service has occurred and immediate action is required.

Maintenance (4)

Color-coded condition: Brown

Indicates that the device has been taken offline for maintenance purposes.

Suppressed (5)

Color-coded condition: Gray

Indicates that the device cannot be reached due to a known error condition that exists on another device.

Initial (6)

Color-coded condition: Blue

Indicates that contact with the device has not yet been established.

Variable

Color-coded condition: N/A; evaluates to a severity that has a color-coded condition

Lets you to assign an alarm severity based on the value of a variable binding. For example, if the value of the variable binding is 1, then the alarm is assigned a severity level of Minor.

To use this option, do the following:

- For Severity, select Variable.
- For Event Variable, specify the ID of the event variable that stores the variable binding value to use to determine the severity level of the alarm. You must specify a variable whose possible values are enumerated and directly correspond to the numeric severity levels used by CA Spectrum (identified in the first column in this table).

Conditional

Color-coded condition: N/A; evaluates to a severity that has a color-coded condition

Lets you to assign an alarm severity based on the value of a variable binding. It also allows you to select the set of enumerated values and corresponding CA Spectrum alarm severity levels to use. This is useful if, for example, the variable binding defines a set of alarm severity levels that differ from those used in CA Spectrum.

To use this option, do the following:

- For Severity, select Conditional.
- For Event Variable, specify the ID of the event variable that stores the variable binding value to use to determine the severity level of the alarm. You must specify a variable whose possible values are enumerated. The actual value in the event variable will be used as the key to look up a corresponding CA Spectrum alarm severity level in a severity mapping file.
- Below the severity level drop-down list, select the file that contains the user-defined mappings of variable binding values and CA Spectrum alarm severity levels.

More information:

[Create Alarm Severity Mappings for the Conditional Severity Level](#) (see page 53)
[Modifying Alarm Severity Mappings for the Conditional Severity Level](#) (see page 55)

[About Event Disposition Files](#) (see page 115)

Enable or Disable Alarms of a Severity Type

Alarms use a large amount of resources such as memory and processing time. CA Spectrum lets you disable an alarm of a severity type to reduce the impact on system performance. The disabled alarm is available internally but you cannot view it in the user interface. The disabled alarm does not support alarm attributes like discriminators. By default, alarm types with severity levels Initial and Suppressed are set as disabled. All other alarm types continue to exist as regular alarms.

Note: Normal severity alarms do not support discriminators.

To disable an alarm of a severity type

1. Select the VNM Model.
2. Click the Information tab and expand the Alarm Management section.
3. Right-click the alarm and select Disable.

The alarm is disabled and the change takes effect immediately. The existing alarms of the selected severity type continue to be regular or internal alarms until they are cleared.

To enable an alarm of a severity type

1. Select the VNM Model.
2. Click the Information tab and expand the Alarm Management section.
3. Right-click the alarm and select Enable.

The alarm is enabled and the change takes effect immediately. The existing alarms of the selected severity type continue to be regular or internal alarms until they are cleared.

Notes:

- Even after enabling alarms, you must verify that the client-side filters are adjusted before you can view the alarm in the client applications.
- We do not recommend enabling the Suppressed alarm type as it can adversely impact performance.

Specify an Alarm Cause Code

An *alarm cause code* is an 8-digit, hexadecimal code that identifies the probable cause of the alarm. When you save the event and associated alarm to a landscape, a mapping between the event code and the associated alarm code is added to the event configuration file that determines how to process the event, referred to as the event disposition file. This mapping is the mechanism by which CA Spectrum identifies if an alarm should be generated for an event, and if so, which one.

As a convention, events that generate alarms typically use their event codes as alarm cause codes, and for this reason, the event code of the event is the default alarm cause code of any alarm. However, you can change the alarm cause code if desired. For example, you might want to change the alarm cause code if you have an existing, more generic alarm that you want to be generated whenever the event occurs.

The alarm cause code is also used to name the underlying probable cause file that contains the alarm-related messages (symptoms, causes, and recommended actions) to display in OneClick when the alarm occurs. Each probable cause file is named Prob<alarm_cause_code>, where <alarm_cause_code> is the code you specify on the Alarms tab in Event Configuration.

To specify an alarm cause code, do one of the following:

- Accept the default code, which readily identifies the alarm with the event that generates it. If the supporting probable cause file does not exist, it will be created automatically by Event Configuration when you save the event and alarm changes to a landscape.
- Click Browse, and in the Select Alarm Cause Code dialog, select an existing code. The displayed list includes all of the alarm cause codes for all loaded events that generate alarms. In this case, the supporting probable cause file will be updated automatically when you save the event and alarm changes to a landscape.
- Enter a new 8-digit, hexadecimal value. If the supporting probable cause file does not exist, it will be created automatically by Event Configuration when you save the event and alarm changes to a landscape.

Note: If you create an alarm-generating event and save it to one or more landscapes, and you later change the alarm cause code to a new code and then save that change to the landscapes, Event Configuration automatically creates a new probable cause file that is named based on the new alarm cause code. However, to remove the probable cause file named using the old code (if it is not used by any other alarm-generating events), you must do so manually.

More information:

- [Saving Events to Landscapes](#) (see page 18)
- [About Event Disposition Files](#) (see page 115)
- [Probable Cause Files](#) (see page 163)
- [Contents of a Probable Cause File](#) (see page 164)

About Specifying Symptoms, Causes, and Recommended Actions

If an event generates an alarm, you need to supply several plain text messages that describe the symptoms, probable causes, and recommended corrective actions for the alarm. These messages are displayed on the Alarm Details tab in OneClick, as shown in the following image.

The screenshot displays the 'Alarm Details' tab in OneClick. The alarm title is 'BAD LINK DETECTED', dated 'Jan 15, 2008 9:06:09 AM EST'. The description reads: 'The link status of port (name - sysName_10802, type - CTMIFPort) is now "bad".' A red icon with the number '10802' is visible in the top left corner. The interface is divided into two main sections. The left section contains configuration options: Severity (Critical), Impact (0), Acknowledged (set), Clearable (Yes), Trouble Ticket ID (set), Assignment, Landscape (server01 (0xf700000)), Status (set), and Web Context URL. The right section contains descriptive text: Symptoms (A port is reporting a BAD link.), Probable Cause (1) Cable is not connected., 2) A backplane interface is broken, disallowing data flow., and Actions (1) Make sure the cable is fastened securely on both ends., 2) Ensure that the cable is not broken., 3) Check backplane of device., followed by a note: '** If the connection on this port is modeled the devices and view is available from the pipe representing the connection.'

The text messages that you enter for the symptoms, probable causes, and recommended corrective actions for an alarm are stored in an alarm configuration file referred to as a probable cause file. The fields that support traps get auto-populated, if the MIB that defines the trap is in the MIB Tools MIB database. For example, when you create a new alarm, the trap specific Alarm Title is auto-populated assuming the MIB in which the trap is defined is found in the MIB database.

More information:

[About Probable Cause Files](#) (see page 163)

Specify Alarm Options

You can specify the following advanced options for an alarm:

Alarm is Persistent

If selected, the alarm is retained in memory if the SpectroSERVER is shut down and then restarted.

Alarm is User Clearable

If selected, the alarm can be cleared by users.

Generate a Unique Alarm for Each Event

If selected, a unique alarm is generated each time that the event occurs. By default, CA Spectrum does not generate a new alarm each time the same event occurs if an alarm already exists for the event on the model; selecting this option changes this default behavior.

Using Event Variable Discriminators to Generate Alarms

By default, if an alarm exists on a given model as a result of an event, CA Spectrum does not generate another alarm if the same event occurs again on the model. This default behavior prevents an event that can occur multiple times due to the same condition. However, often you want multiple alarms to occur if the conditions of the event change, and you can specify this type of behavior using event variable discriminators.

Event variable discriminators are numeric values that refer to the IDs of the event variables in an event. In turn, the event variable IDs are mapped (in the AlertMap file) to the OIDs of the variable bindings sent with the trap. The discriminators let CA Spectrum to determine whether to generate alarms for multiple instances of the same event based on the values of the variable bindings sent with the traps.

If you configure an event to generate an alarm, and you specify one or more discriminators, the alarm is generated if the values of the referenced event variables are *different*. In this way, you can specify that alarms should be generated for distinct event instances (which have the same event code) even if an alarm already exists on the model.

As an example, assume you have configured event 0x3b10011 to generate alarm 0x3b10011, and you have specified that event variables 1 and 3 are event variable discriminators. This configuration means that, if an existing alarm 0x3b10011 already exists on the model, another alarm is not generated unless the values for *both* event variables 1 and 3 are *different* in the new event instance when compared to current alarms on the model generated from the same event.

When you are configuring an alarm, specify the event variable discriminators by entering a comma-separated list of IDs, for example:

1,3,5

You must enter each ID; ranges of IDs are not supported.

Note: Event discriminators cannot be specified for normal, maintenance, suppressed, or initial severity alarms.

Creating Dynamic Alarm Title

The alarm title is taken from the PCause files (\$SPECROOT/SG-Support/CsPCause).

In OneClick, whenever the dynamic alarm title attribute has any value, the dynamic string is displayed instead of the static alarm title from the probable cause file. The static alarm title will be displayed by default, if there is no value in the dynamic alarm title attribute.

You can use the dynamic alarm title variable to create a dynamic alarm title. The dynamic varbind id is 76620 (or 0x12b4c). Once you set the dynamic alarm title variable, you see more information about the alarm in the title. To use the functionality, you can create an event which has that varbind set to any value e.g. by mapping a trap variable to that ID in an alert map file. You can also copy some other event variable via an event rule, and then map the event to an alarm using that ID as a discriminator.

Example: Create a Dynamic Alarm Title

You can either create 0x050e1106 through an alert map or through the condition rule. This example maps an alert to an event, copying over one alert variable (1.3.6.1.10.1.2) as a dynamic alarm title. The dynamic varbind id is 76620 (or 0x12b4c).

```
1.3.6.1.4.4.1.6.3 0x050e1106 1.3.6.1.4.1.10.1.1(1,0)\
1.3.6.1.4.1.10.1.2(76620,0)
```

You can also use an event rule to copy over a varbind (here ID 7 from event 0xffff0000) to be used as dynamic title id in the new event (0x050e1106).

```
0xffff0000 E 50 R CA.EventCondition, "default", "0x050e1106 7:76620"
```

For both cases, you can create a minor alarm which will show the dynamic title, discriminating on the dynamic value ID. This shows an individual alarm for each value.

```
0x050e1106 E 50 A 1,0x050e1106,76620
```

Example: Clear a Dynamic Alarm Title

This example shows how to clear a dynamic alarm title using the Dynamic ID value.

```
0x050e1107 E 50 C 0x050e1106,76620
```

Note: You do not have to use the dynamic alarm title attribute as discriminator. In this case, there will only be one alarm, with one title, with the value from the first event which created the alarm. All subsequent events will just be attached to that alarm, even when you have different dynamic title values. If they use the discriminator, then of course they get one alarm per title value. This uses the regular discriminator feature.

More information:

[Using Event Variable Discriminators to Generate Alarms](#) (see page 50)

Configuring Alarm Severity Mappings for the Conditional Severity Level

A *severity mapping file* is an ASCII file that is used to determine the actual severity of an alarm instance to generate when the alarm configuration specifies an alarm severity of Conditional.

The *Conditional alarm severity* lets you assign the alarm severity based on the value of a variable binding sent with the trap that triggered the event. Moreover, it also lets you select the set of mappings between values and alarm severity levels to use when determining the actual alarm severity.

It is the severity mapping file that defines the mappings between possible variable binding values and CA Spectrum alarm severity levels. You can create severity mapping files and customize those that are provided by default with CA Spectrum.

More information:

[Create Alarm Severity Mappings for the Conditional Severity Level](#) (see page 53)
[Modifying Alarm Severity Mappings for the Conditional Severity Level](#) (see page 55)

Create Alarm Severity Mappings for the Conditional Severity Level

If you create a severity mapping file, it is installed in the following location when you save the change to a landscape:

```
<$SPECROOT>/custom/Events/<vendor_directory>/SeverityMaps/<file name>
```

where <vendor_directory> is the directory and <file_name> is the file name that you specified when you created the mapping file using Event Configuration. If you do *not* specify a directory when you create the file, it is installed in the following folder instead:

```
<$SPECROOT>/custom/Events/CA/SeverityMaps/<file name>
```

Important! Custom severity mapping files override those provided with CA Spectrum, the latter of which are located in <\$SPECROOT>/SS/CsVendor/<vendor_directory>/SeverityMaps.

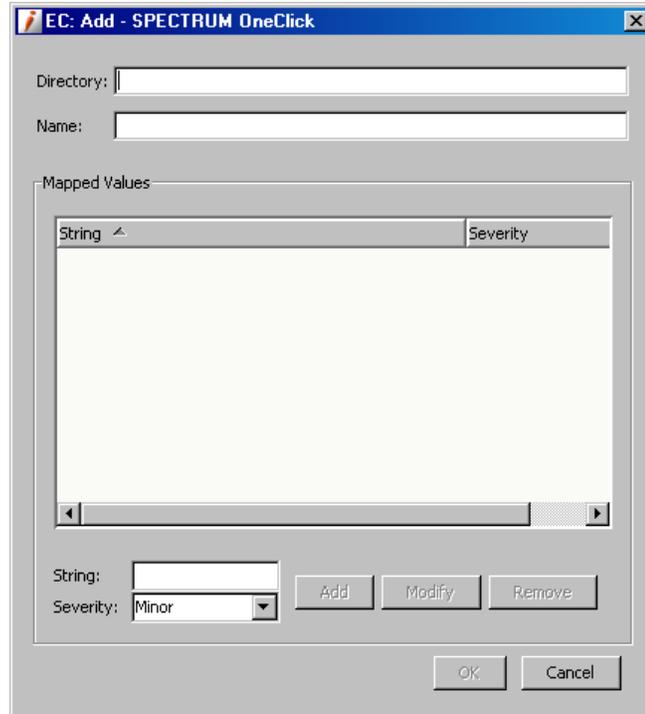
To create an alarm severity mapping file

1. Display the event that generates the alarm, and click the Alarms tab.
2. If you have not already done so, do the following:
 - a. For Severity, select Conditional.
 - b. For Event Variable, enter or select the event variable that contains the variable binding value to use to determine the alarm severity.
3. Select any severity mapping file and click Configure.

The Configure Conditional Alarm Severity dialog opens, displaying the contents of the selected severity mapping file.

4. Click .

The severity mapping file add dialog opens.



5. Complete the fields as follows:

Directory

Specifies the directory in which to save the severity mapping file.

Name

Specifies a name for the severity mapping file; this should be the name of the variable binding whose values are enumerated in the file.

6. Add the mappings between the string representations of the variable binding values and CA Spectrum alarm severity levels (described in [Specifying an Alarm Severity](#) (see page 45)) as follows:
- To add a mapping, enter a unique string value for String, select a severity for Severity, and click Add.
If you want the alarm to be cleared—instead of generated—if the corresponding variable binding value is sent, select Clear.
 - To change a mapping, select the mapping, change the value for String or for Severity (or both), and click Modify.
 - To remove a mapping, select the mapping, and click Remove.
7. Click OK twice.

Modifying Alarm Severity Mappings for the Conditional Severity Level

You can make changes to your custom severity mapping files and those provided by default with CA Spectrum at any time.

If you modify a severity mapping file that was provided with CA Spectrum, a customized version of the file is installed in the following location when you save the change to a landscape:

```
<$SPECROOT>/custom/Events/<vendor_directory>/SeverityMaps/<file name>
```

Important! Custom severity mapping files override those provided with CA Spectrum, the latter of which are located in `<$SPECROOT>/SS/CsVendor/<vendor_directory>/SeverityMaps`.

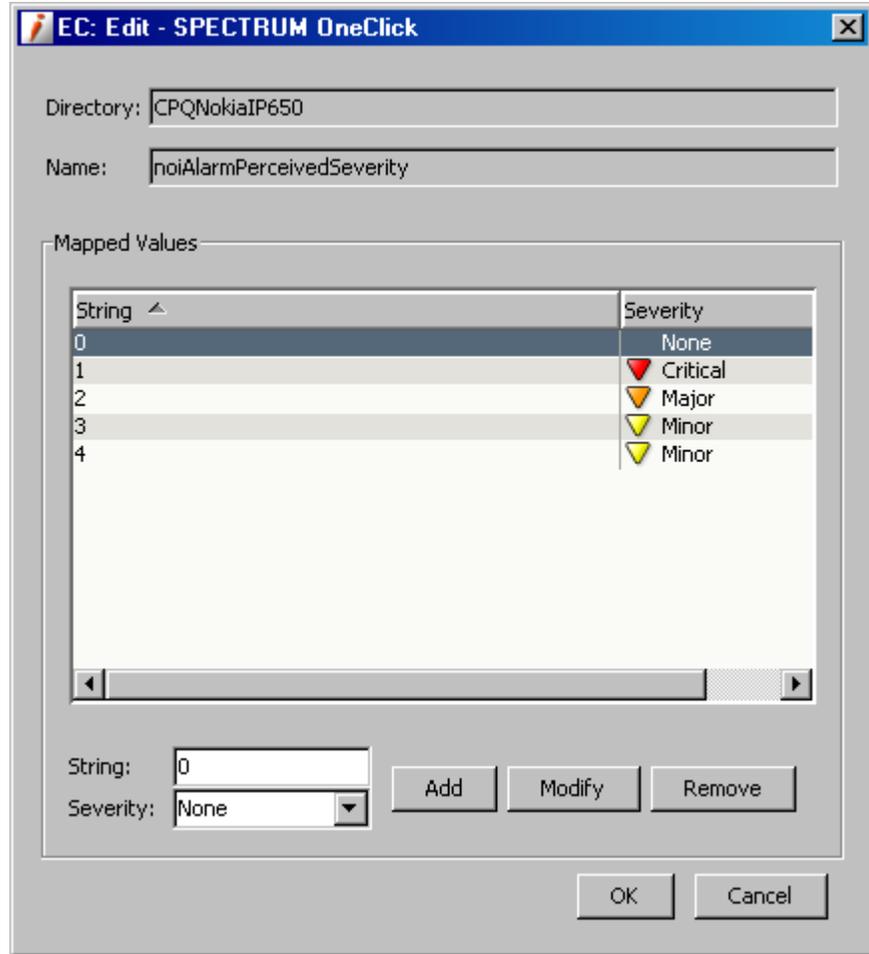
To modify an alarm severity mapping file

1. Display the event that generates the alarm, and click the Alarms tab.
2. If you have not already done so, do the following:
 - a. For Severity, select Conditional.
 - b. For Event Variable, enter or select the event variable that contains the variable binding value to use to determine the alarm severity.
3. Select the severity mapping file to use to determine the actual severity level of the alarm, and click Configure.

The Configure Conditional Alarm Severity dialog opens, displaying the contents of the selected severity mapping file.

4. Click .

The severity mapping file edit dialog opens.



5. Modify the mappings between the string representations of the variable binding values and CA Spectrum alarm severity levels as follows:
- To change a mapping, select the mapping, change the value for String or for Severity (or both), and click Modify.
If you want the alarm to be cleared—instead of generated—if the corresponding variable binding value is sent, select Clear.
 - To add a mapping, enter a unique string value for String, select a severity for Severity, and click Add.
 - To remove a mapping, select the mapping, and click Remove.
6. Click OK twice.

Configure Events to Clear Alarms

You can specify that the currently selected event should clear one or more alarms using the Alarms tab in the Details panel, which is shown in the following image.

Contents: Event 0x10618

Event Message
Port violation reset. Port {I 3}{Instance ID {O 4}} on board in slot {I 1}{Instance ID {O 2}} of {t} (name - {m}) has been reset. (Trap type : 0x0118)

Details

Alarms | Event Rules | Event Options

Generated Alarm

Severity: Cause Code: Browse... Event Variable Discriminators:

Type:

Symptoms | Probable Causes | Recommended Actions | Alarm Options

Alarm is Persistent
 Alarm is User Clearable
 Generate a Unique Alarm for Each Event

Cleared Alarm(s)

Cause Code	Type	Event Variable Discriminators	All Alarms
0x10617	PORT SECURITY VIOLATION TRAP RECEIVED	1,3	

Filter: Displaying 1 of 1

When you configure an event to clear alarms and then save that change to a landscape, you create a mapping between the event and the alarms to be cleared in a configuration file referred to as an event disposition file.

To configure an event to clear alarms

1. Select the event in the Navigation panel, and then click the Alarms tab in the Details panel.

2. Under Cleared Alarm(s), click  (Adds an alarm to the list).

The Add Cleared Alarm dialog opens displaying the alarm cause codes for all alarms currently loaded into Event Configuration.

3. Click Browse, and in the Select Alarm Cause Code dialog, select the alarm cause code of the alarm to clear.

To help you identify the desired alarm, you can enter a text string in the Filter text box to filter the list to include only the alarms with displayed properties that contain the text string.

4. Click OK.
5. If the alarm you want to clear was generated based on event variable discriminators, select Clear Options, and then specify how the alarm should be cleared. You can select one of the following options:

All Alarms

Select this option if the event should clear all existing instances of the alarm regardless of whether the values in the alarm-clearing event match the values stored in the alarm instances.

Event Variable Discriminators

Select this option if the event should clear existing instances of the alarm based on the values of the alarm's variables (which are copied from the alarm-generating event to the alarm when the alarm is generated). Then enter a comma-separated list of event variable IDs (for example: 1,3,5). You must enter each ID; ranges of IDs are not supported.

The alarm is cleared by the alarm-clearing event if the values in event match the values stored in the alarm.

For examples of using event discriminators to clear alarms, see [Clearing Alarms](#) (see page 124).

6. Click OK.
7. (Optional) Repeat the preceding steps to add additional alarms to be cleared.

More information:

[About Event Disposition Files](#) (see page 115)

Modify Events

You cannot delete the events that are authored by CA and provided with CA Spectrum. However, you can customize them to meet your requirements just as you would any other events.

In addition, you can undo any customizations that you make to a CA-authored event by deleting the event. *For CA-authored events only*, this action does not delete the event but instead reverts it to its default configuration.

When you delete (revert) a CA-authored event, the author of the event changes from "Custom" back to "CA." To identify the author of an event, add the Author column to the table of events in the Navigation panel, as described in [Adding and Removing Columns from the Events Table](#) (see page 25).

To modify events

1. Select the event in the Navigation panel.

The event details are displayed in Contents and Details panels.

2. Modify the event details.

The event is displayed in bold. The Modification column displays *Edited*.

Note: The event is marked *Edited*. However, the event is not saved. You can make more updates such as create more events, modify existing events, or delete events, and save all the updates at one time.

3. Do *one* of the following:

- a. Go to File, Save All.

All marked events in the Navigation panel are saved.

- b. Select events in the Navigation panel, and go to File, Save Selected.

Note: You can select multiple events using the Shift key.

The selected events are saved.

The events are modified. The events appear in the normal font and are not marked in the Modification column.

Delete Custom Events

You can delete any event that was not authored by CA. To identify the author of an event, add the Author column to the table of events in the Navigation panel.

Important! Do not delete an event unless you are certain it is no longer required. If you delete an event that generates a needed alarm, CA Spectrum will be unable to inform you about a problem in the network infrastructure.

To delete events

1. Select the event to delete in the Navigation Panel, and click  (delete).

The event is displayed in italics. The Modification column displays *Deleted*.

Note: The event is marked *Deleted*. However, the event is not deleted until you save the changes. You can make more updates such as create more events, modify existing events, or delete events, and save all the updates at one time.

2. Do one of the following:

- a. Go to File, Save All.

All marked events in the Navigation panel are saved.

- b. Select events in the Navigation panel, and go to File, Save Selected.

Note: You can select multiple events using the Shift key.

The selected events are saved.

The events are deleted. The events appear in the normal font and are not marked in the Modification column.

More information:

[Add and Remove Columns from the Events Table](#) (see page 25)

Chapter 4: Working with Event Rules

This section contains the following topics:

[Event Rules](#) (see page 61)

[Create Event Rules](#) (see page 65)

[Modifying Event Rules](#) (see page 92)

[Delete Event Rules](#) (see page 93)

Event Rules

Event rules lets you specify evaluation logic that determines how CA Spectrum processes the events to which you are applying the rules and what actions to perform in response to the events.

An event rule stipulates the conditions under which an event condition, a pattern of events, or a combination of events in a particular order or over a particular period of time sets the generation of another event.

You can create multiple rules for a single event.

Event rules are stored in event configuration files referred to as event disposition files.

More information:

[Using Procedures in Event Processing](#) (see page 95)

Event Condition Rules

An *event condition rule* creates an event when specific conditions are satisfied. The input to the rule is a list of conditions and associated events. Each condition is evaluated until one is found that evaluates to TRUE, and then the corresponding event is created.

The following are the parameters to the rule:

- The event that starts the evaluation of the rule
- A conditional expression to evaluate
- The event to generate if the conditional expression is true

An event condition rule can include multiple conditional expressions and corresponding output events.

More information:

[Configuring Event Condition Rule Settings](#) (see page 66)

Event Pair Rules

In some cases, you expect events to happen in pairs, and if the second event does not occur, this may indicate a problem in the computing infrastructure. An *event pair rule* creates an event based on this scenario. If the first of two expected events is generated but the second event does not follow the first, a new event is generated in response. You can specify the amount of wait time that can elapse before the new event is generated.

Note: Other unrelated events can be generated between the first and the second event. They do not affect execution of the rule.

More information:

[Configuring Event Pair Rule Settings](#) (see page 70)

Event Rate Rules

Some types of events do not indicate a problem unless the frequency at which they are generated reaches a specific threshold within a specific amount of time. An *event rate rule* creates an event based on this scenario. When a number of events of the same type that is, with the same event code are created within a given time period, a new event is created in response.

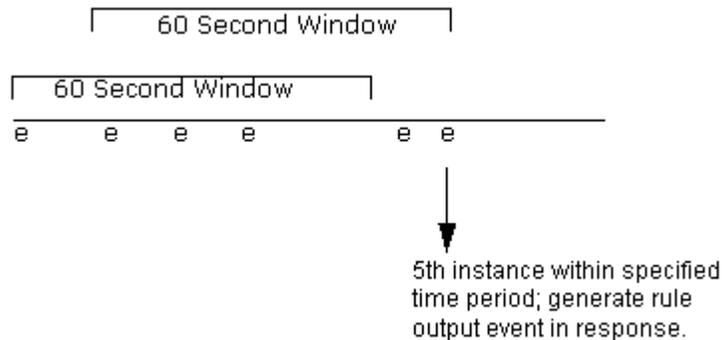
Note: Other, unrelated events do not affect execution of the rule.

Event rate rules never terminate. Once the conditions of the rule are met and a new event is created in response, the rule remains active, but no additional event is created as long as the frequency at which the evaluated events remains at or above the specified rate in the rule. If the frequency drops below the specified rate, and then subsequently exceeds that rate again, another new event is generated in response.

An event rate rule can use either of the following methods to define the window of time in which the events must occur:

Sliding Window: When the rule uses this type of time window, if the specified number of events (or more) ever occurs in any window of the specified time period, the output event is created in response. This type of time window is best suited for accurately detecting a short burst of events.

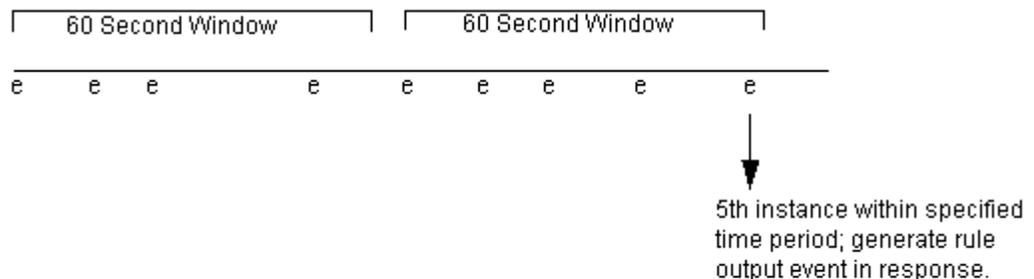
For example, the following illustration shows the sliding time windows that are active for a rule that watches for five instances of a given event (e) within a specified time period.



When a sliding time window is used for a rule, if the rule generates a rule output event, all active time windows are terminated, and a new time window automatically begins.

Sequential Window: When the rule uses this type of time window, non-overlapping time windows are examined, one after another, to determine if the requisite number of events has occurred within the time window. This type of time window is best suited for detecting a long, sustained train of events.

For example, the following illustration shows the sequential time windows that are opened and closed for a rule that watches for five instances of a given event (e) within a specified time period.



If the current time window closes due to time period expiration, or if the rule creates an output event in response, the window is not opened next time until a new event occurrence is detected.

More information:

[Configuring Event Rate Rule Settings](#) (see page 73)

Event Series Rules

An *event series rule* creates a new event when a given event is followed by one or more other events in an ordered or unordered sequence. The combination of events that must occur can include any number and type of event, and you can specify the amount of wait time that can elapse during which the sequence of events must occur.

Note: Other, unrelated events can be generated during the wait time. They do not affect execution of the rule.

More information:

[Configuring Event Series Rule Settings](#) (see page 76)

Event Counter Rules

An *Event Counter rule* counts events. It watches for two events, one increasing the count, the other one decreasing it again. An event is generated whenever the count is higher than a threshold, and also once it falls below the set threshold. The EventCounter rule remains instantiated and counts events. It does not terminate.

Heartbeat Rules

The *Heartbeat rule* watches for a periodic heartbeat event, and generates a new event when the heartbeat event is missing. There is a separate event to stop the rule instance, if required.

Single Event Rules

The *Single Event rule* watches for a single occurrence of a target event. The SingleEvent rule reduces an event stream where one event ('up' event) may occur multiple times, before a reset ('down') event is seen. Instead of the multiple 'up' events, a single event is set that can be reused in other rules, denoting the condition ('up' or 'down'). An event is generated the first time the target event is seen. The rule does not trigger unless the clearing event is seen again. It will then reset the behavior to the initial state. If needed, a separate event can be generated when the clearing event is seen.

Solo Event Rules

The *SoloEvent* rule finds an instance of the target event that is not followed by or preceded by any other user defined event in a defined time window. The time periods are configurable, and there is a separate event to stop the rule.

User-Defined Event Rules

Most of the CA-shipped event rules are supported on the Event Configuration User Interface. However, in some cases, CA can create a special rule or customize a rule for an individual customer. A customer can create a rule as well.

Event disposition entries using such rules are not supported by the Event Configuration Editor. The event rule displayed is read-only. The rule entry must be edited in a text editor and reloaded in the event disposition files. Such custom event rule entries need to have the correct vendor code and rule name, depending on which vendor supplied the rule. For example, a custom rule entry for a rule called 'MyOwnRule' from vendor 'MyOwnVendor' will have the following syntax:

```
0xffff00002 E 50 R MyOwnVendor.MyOwnRule, <rule parameters>
```

Note: The xml rule definition file is located at
\$SPECROOT/SS/CsVendor/<Vendor Name>/EventRules/<RuleName>.xml.

Create Event Rules

Before creating an event rule, we recommend that you start the application and examine the rules that are provided for various events created by CA.

You can create an event rule in two ways:

- From scratch.
- By copying an existing event rule and modifying the copy. This method is only useful if the new rule will also be triggered by the same event that triggers the rule you want to copy.

Note: Event rule definitions are stored in event configuration files referred to as event disposition files.

To create an event rule from scratch

1. In the Navigation panel, select the event that will activate (trigger) the event rule.
2. In the Details panel, click the Event Rules tab.
The list of event rules for the selected event appears.
3. Click  (list) and select the type of event to create from the drop-down list.
The configuration dialog for the selected type of event rule opens.
4. Configure the event rule as required.
5. Click OK to save the event rule to the event.

To create an event rule from a copy

1. In the Navigation panel, select the event that has the event rule that you want to copy.
2. In the Details panel, click the Event Rules tab.
The list of event rules for the selected event appears.
3. Select the event rule to copy and click  (copy).
The event rule configuration dialog opens.
4. Modify the configuration of the event rule as required.
5. Click OK to save the event rule to the event.

More information:

- [Configuring Event Condition Rule Settings](#) (see page 66)
- [Configuring Event Pair Rule Settings](#) (see page 70)
- [Configuring Event Rate Rule Settings](#) (see page 73)
- [Configuring Event Series Rule Settings](#) (see page 76)

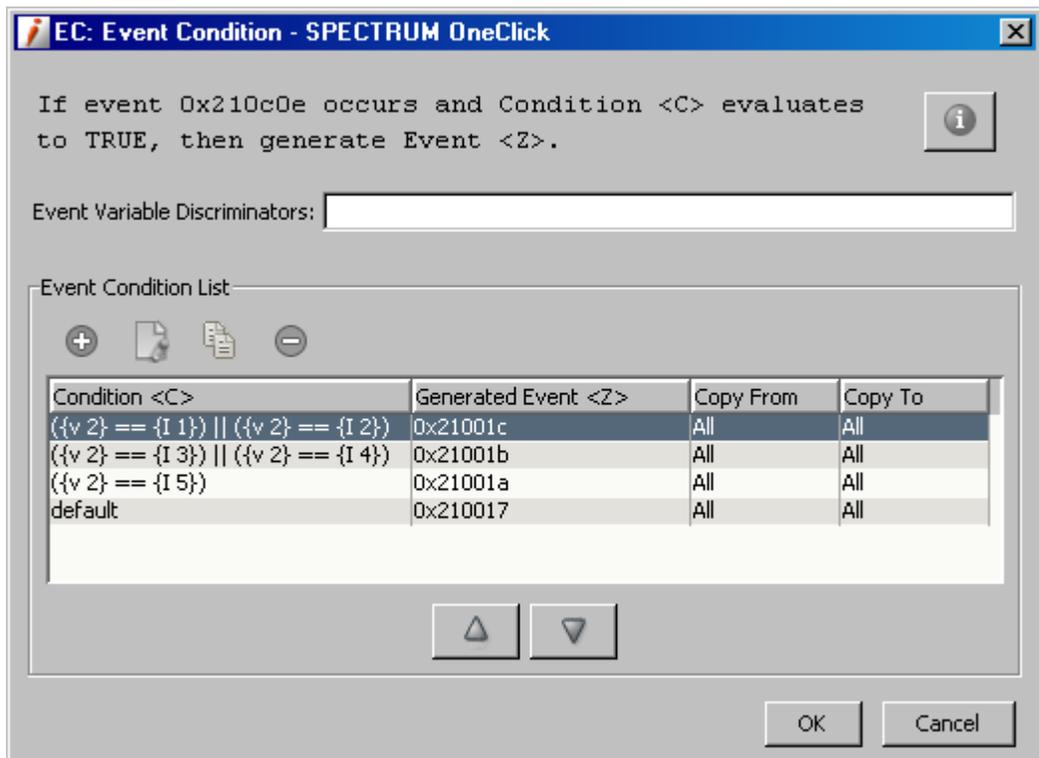
Configuring Event Condition Rule Settings

To create an event condition rule you must create one or more conditions against which the event for which you are creating the rule is evaluated. For each condition, you must also specify an event that CA Spectrum generates if the given condition is met. Optionally, you can also specify that one or more values of the event that triggered the rule should be copied to the rule output event.

As an example, assume you want to create the following event condition rule (expressed in pseudo-code) for event 0x210c0e:

If event 0x210c0e occurs, evaluate the following:
 if event variable 2 = 2 or event variable 2 = 2, then generate 0x0021001c,
 else if event variable 2 = 3 or event variable 2 = 4, then generate 0x0021001b,
 else if event variable 2 = 5, then generate 0x0021001a,
 else, generate 0x00210017.

This rule requires the four conditional expressions shown in the following rule configuration dialog.

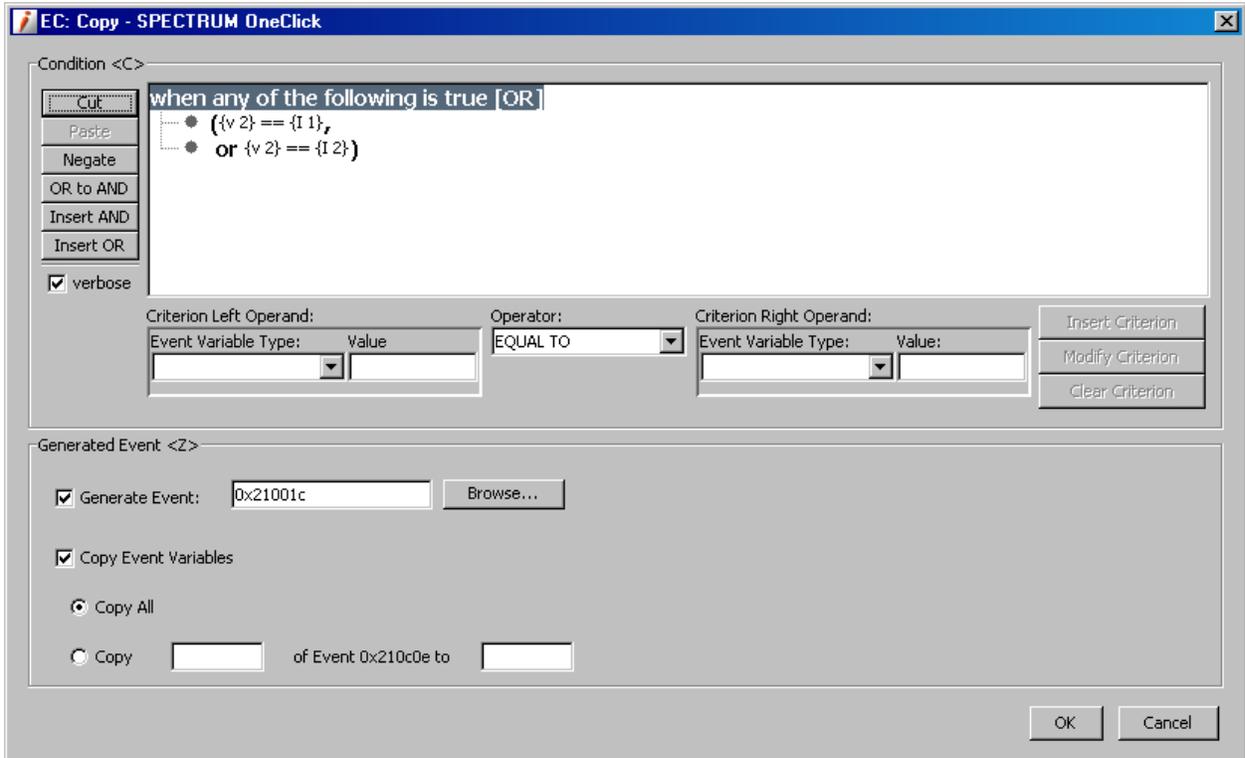


In the dialog, use the buttons above the list of conditions to add new conditions and to modify, copy, and delete a selected condition. Use the up and down arrows below the list of conditions to move a selected condition up or down to change the sequence in which the conditions are evaluated.

The rule output event (generated event <Z>) that corresponds to the first condition that evaluates to TRUE is generated.

Note: Event variable discriminators are a general feature available for all types of rules. However, while you can specify discriminators for an event condition rule, a rule of this type currently does not use them during rule processing.

When you add or modify a condition, you define the conditional expression using the following dialog.



The settings in the dialog include the following:

Condition <C>

The condition to evaluate. Use the controls in the top section of the dialog to create the expression.

For information on conditional expression syntax, see [EventCondition Rule](#) (see page 61).

Generated Event <Z>- Generate Event

The event code of the event to create in response if the associated condition evaluates to TRUE.

To specify the event code, click Browse, select the event in the Select Event dialog (which displays all of the events loaded into Event Configuration), and click OK.

Generated Event <Z> - Copy Event Variables

If you want to copy the values of any event variables in the event that activated the rule to generated event <Z>, select Copy Event Variables, and do one of the following:

- Select Copy All to copy the values of all of the event variables into generated event <Z>.
- Select Copy, and specify a comma-separated list of specific variable IDs whose values should be copied. Ranges of IDs are also supported, for example, "1-4".

Note: For information on the proper syntax to use when specifying variable IDs or ranges of IDs using Event Configuration, see [Copying Variable Values from Contributing Events to the Rule Output Event](#) (see page 91).

Note: To create a default condition that always evaluates to TRUE, in the condition configuration dialog, simply select DEFAULT for Operator, and click Insert Criterion.

Configuring Event Pair Rule Settings

You configure an event pair rule using the following dialog.

EC: Event Pair - SPECTRUM OneClick

If event 0x11c003c occurs and is NOT followed by Event <Y> within <T> seconds, then generate Event <Z>.

Event Variable Discriminators:

Event <Y>

Event Code:

Time <T>

Time Interval: Days +

Generated Event <Z>

Generate Event:

Copy Event Variables

Copy All

Copy of Event 0x11c003c to

The settings in the dialog include the following:

Event Variable Discriminators

A comma-separated list of the IDs of the event variables in the event to evaluate as a part of the rule.

Note: You must enter each ID; ranges of IDs are not supported.

Event <Y> - Event Code

The event code of the event that, within the specified time period, should follow the event that activates (triggers) the rule. Event <Y> is the second event in the pair of events.

To specify the event code, click *Browse*, select the event in the *Select Event* dialog (which displays all of the events loaded into *Event Configuration*), and click *OK*.

Time <T> - Time Interval

The period of time that begins when the first event is created and during which the second event (event <Y>) should occur.

Generated Event <Z> - Generate Event

The event code of the event to create in response if the second event (event <Y>) does not follow the first event within the specified time interval.

To specify the event code, click *Browse*, select the event in the *Select Event* dialog (which displays all of the events loaded into *Event Configuration*), and click *OK*.

Generated Event <Z> - Copy Event Variables

If you want to copy the values of any event variables in the first event that activated the rule to generated event <Z>, select *Copy Event Variables*, and do one of the following:

- Select *Copy All* to copy the values of all of the variables into generated event <Z>.
- Select *Copy*, and specify a comma-separated list of specific variable IDs whose values should be copied. Ranges of IDs are also supported, for example, "1-4".

Note: For information on the proper syntax to use when specifying variable IDs or ranges of IDs using *Event Configuration*, see [Copying Variable Values from Contributing Events to the Rule Output Event](#) (see page 91).

Event Variable Discriminators in Event Pair Rules

The use of discriminators not only lets multiple instances of the rule to be generated due to the same event, but also affects rule processing and termination. More specifically, if discriminators are used, the following occurs:

When the event is generated, if there are active instances of the rule, another instance of the rule is created only if the values of the specified variables are different in the new event when compared to the events that activated the existing rule instances. (If you do not specify any discriminators for a rule, if a single instance of the rule is active, subsequent instances of the rule are never generated when subsequent instances of the event occur.)

To terminate the rule without generating the rule output event, the values of all of the specified event variables must be the same in both the first event (that triggered the rule) and the second event (for which the rule is watching) in the pair.

As an example, assume you create an event pair rule that watches for a port down event followed by a port up event, and you specify the event variable that stores the interface ID of the port as an event variable discriminator. As a result, if device A has ports 1 and 2, and port 1 goes down, a rule instance for the event is generated. If port 2 subsequently goes down, another rule instance is also generated due to the same event because the interface ID in the second instance of the event is different. There are now 2 active instances of the rule due to port down events related to 2 different ports.

To continue the example, if port 2 goes up within the specified time period, the associated rule will immediately terminate without generating the rule output event if the port up event that is generated stores the interface ID of port 2 in the same event variable (that is, the IDs of the event variables in the first event and second event in the pair that store the interface ID are the same). If a similar event is not generated for port 1 within the specified time period, the associated rule will terminate by generating the rule output event.

Note: To use event variable discriminators effectively in an event pair rule, the IDs of the event variables in the contributing events must match. That is, the same event variables must store the same variable binding data. If this is not the case, you can create an event condition rule that is activated by the first event and that copies the data to event variables that have the appropriate IDs in the rule output event; you can then use the rule output event as the second event in the pair.

Configuring Event Rate Rule Settings

You configure an event rate rule using the following dialog.

EC: Event Rate - SPECTRUM OneClick

If event 0x210b94 occurs <N> time(s) within Time Interval <T> seconds, then generate Event <Z>.

Event Variable Discriminators: 8

Event Rate

Occurrences <N>: 3

Time <T>: 0 Days + 00:07:00

Sliding Window

Generated Event <Z>

Generate Event: 0x00210b95 Browse...

Copy Event Variables

Copy All

Copy [] of Event 0x210b94 to []

OK Cancel

The settings in the dialog include the following:

Event Variable Discriminators

A comma-separated list of the IDs of the event variables in the event to evaluate as a part of the rule.

You must enter each ID; ranges of IDs are not supported.

The use of discriminators not only lets multiple instances of the rule to be generated due to the same event, but also affects rule processing. More specifically, if discriminators are used, the following occurs:

- When the event is generated, if there are active instances of the rule, another instance of the rule is created only if the values of the specified variables are different in the new event when compared to the events that activated the existing rule instances. (If you do not specify any discriminators for a rule, if a single instance of the rule is active, subsequent instances of the rule are never generated when subsequent instances of the event occur.)
- To generate the rule output event, the values of the variables must be the same in all instances of the same event (and all other rule conditions must be met).

Event Rate - Occurrences <N>

The number of instances of the same event that must be created within the specified time period in order for the rule to generate the output event.

Event Rate - Time <T>

The period of time during which <N> occurrences of the same event must occur in order for the rule to generate the output event.

Event Rate - Sliding Window

The type of time window to use. Select Sliding Window to use a *sliding window* of time; clear Sliding Window to use a *sequential window* of time. For a description of both, see [Event Rate Rules](#) (see page 62).

Generated Event <Z> - Generate Event

The event code of the event to create in response if <N> occurrences of the same event occur within the specified period of time.

To specify the event code, click Browse, select the event in the Select Event dialog (which displays all of the events loaded into Event Configuration), and click OK.

Generated Event - Copy Event Variables

If you want to copy the values of any event variables in the event that activated the rule to generated event <Z>, select Copy Event Variables and do one of the following:

- Select Copy All to copy the values of all of the event variables into generated event <Z>.
- Select Copy, and specify a comma-separated list of specific variable IDs whose values should be copied. Ranges of IDs are also supported, for example, "1-4".

Note: For information on the proper syntax to use when specifying variable IDs or ranges of IDs using Event Configuration, see [Copying Variable Values from Contributing Events to the Rule Output Event](#) (see page 91).

Note: If you want an event rate rule to also generate a different rule output event when the rate at which the trigger event occurs drops below the specified threshold, you can manually modify the rule in the event disposition file and specify that rule output event. For more information, see [EventRateWindow Rule](#) (see page 131).

Configuring Event Series Rule Settings

You configure an event series rule using the following dialog.

EC: Event Series - SPECTRUM OneClick

If event 0x220002 occurs and is followed by Events <S> within window Time Interval <T> seconds, then generate Event <Z>.

Event Variable Discriminators: 1

Events <S>

Event List: Ordered

Event Code	Copy From	Copy To
0x220001	None	None

Time <T>

Time Interval: 0 Days + 00:01:00

Generated Event <Z>

Generate Event: 0x00220006

Copy Event Variables

Copy All

Copy 1,3-6 of Event 0x220002 to 1,3-6

The settings in the dialog include the following:

Event Variable Discriminators

A comma-separated list of the IDs of the event variables in the event to evaluate as a part of the rule.

You must enter each ID; ranges of IDs are not supported.

The use of discriminators not only lets multiple instances of the rule to be generated due to the same event, but also affects rule processing and termination. More specifically, if discriminators are used, the following occurs:

- When the event is generated, if there are active instances of the rule, another instance of the rule is created only if the values of the specified variables are different in the new event when compared to the events that activated the existing rule instances. (If you do not specify any discriminators for a rule, if a single instance of the rule is active, subsequent instances of the rule are never generated when subsequent instances of the event occur.)
- To generate the rule output event, the values of the variables must be the same in all contributing events (that is, in the event that activated the rule and in all events specified in the series).

To use event variable discriminators effectively in an event series rule, the IDs of the event variables in the contributing events must match. That is, the same event variables must store the same variable binding data. If this is not the case, you can create an event condition rule that is activated by a given event and that copies the data to event variables that have the appropriate IDs in the rule output event; you can then use the rule output event in the series.

Events <S> - Event List

Select Ordered if the list (series) of events must occur in a specific sequence to trigger creation of the response event. Select Not Ordered if the events in the series can occur in any order.

Events <S> - Event Code

The series of events that must follow the event that activated the rule in order for the rule to generate the output event. If the events in the series must occur in a specific order, list them in that order.

Use the buttons above the table to add, modify, copy, and remove events from the list. To move an event up or down in the list (if necessary), select the event, and click the up or down arrow.

When you add an event to the series, you can also specify that one or more of the values in its event variables should be copied to the rule output event (generated event <Z>). To do so, in the dialog in which you select the event to add to the series, do one of the following:

- Select Copy All to copy the values of all of the event variables to the rule output event.
- Select Copy, and specify a comma-separated list of specific variable IDs whose values should be copied. Ranges of IDs are also supported, for example, "1-4".

Time <T> - Time Interval

The period of time during which the series of events must occur in order for the rule to generate the output event.

Generated Event <Z> - Generate Event

The event code of the event to create in response if the series of events occurs within the specified period of time and, if required, in the specified order.

To specify the event code, click Browse, select the event in the Select Event dialog (which displays all of the events loaded into Event Configuration), and click OK.

Generated Event <Z> - Copy Event Variables

If you want to copy the values of any event variables in the event that first activated the rule to generated event <Z>, select Copy Event Variables, and do one of the following:

- Select Copy All to copy the values of all of the event variables into generated event <Z>.
- Select Copy, and specify a comma-separated list of specific variable IDs whose values should be copied. Ranges of IDs are also supported, for example, "1-4".

Note: For information on the proper syntax to use when specifying variable IDs or ranges of IDs using Event Configuration, see [Copying Variable Values from Contributing Events to the Rule Output Event](#) (see page 91).

In some cases, you might want an event to be generated in response to a specific series of events, but one of several events might first trigger that same series. To satisfy this scenario, create the same event series rule for all events that will trigger the series.

Alternatively, you might want to watch for any combination of a series of events and in any order, inclusive of the event that triggers the series. To satisfy this scenario, you might create the following rules:

Event that triggers the rule (event to which rule is applied)	Events in the series (not ordered)	Rule output event (generated event <Z>)
0x0001002a	0x0001002b, 0x0001002c	0x0001002f
0x0001002b	0x0001002a, 0x0001002c	0x0001002f
0x0001002c	0x0001002a, 0x0001002b	0x0001002f

Using this set of rules, any combination of events 0x0001002a, 0x0001002b, and 0x0001002c occurring in any order generates event 0x0001002f.

Configuring Event Counter Rule Settings

You configure an event counter rule using the following dialog:

The dialog box is titled "EC: Event Counter - SPECTRUM OneClick". It contains the following sections:

- Count events, up event 0x10002, down event <D>, count threshold <T>, threshold violated event <V>, threshold reset event <R>**: A text area with a help icon (i) on the right.
- Event Variable Discriminators:** A text input field.
- Count down event <D>**:
 - Event Code: [] [Browse...]
 - Copy Event Variables
 - Copy All
 - Copy [] of Event to []
- Counter Occurrence Threshold <N>**:
 - Counter Occurrence Threshold <N>: [3] []
- Threshold violated event <V>**:
 - Event Code: [] [Browse...]
 - Copy Event Variables
 - Copy All
 - Copy [] of Event 0x10002 to []
- Threshold reset event <R>**:
 - Event Code: [] [Browse...]
 - Copy Event Variables
 - Copy All
 - Copy [] of Event 0x10002 to []
- Buttons:** [OK] [Cancel]

The settings in the dialog include the following:

The event for which the rule is defined is the one that counts up the event. The first event initiates a new rule instance and counts (count will be 1).

Event Variable Discriminators

A comma-separated list of the IDs of the event variables in the event to evaluate as a part of the rule.

Note: You must enter each ID; ranges of IDs are not supported. Event variable discriminators are a general feature available for all types of rules.

Count down event <D> - Event Code

Sets the event code that counts down by one. To specify the event code, click *Browse*, select the event in the *Select Event* dialog which displays all of the events loaded into *Event Configuration*, and click *OK*.

Count down event <D> - Copy Event Variables

If you want to copy the values of any event variables in the first event that activated the rule to generated event <D>, select *Copy Event Variables*, and do one of the following:

- Select *Copy All* to copy the values of all of the variables into *Count down event <D>*.
- Select *Copy*, and specify a comma-separated list of specific variable IDs whose values should be copied. Ranges of IDs are also supported, for example, "1-4".

Note: For information on the proper syntax to use when specifying variable IDs or ranges of IDs using *Event Configuration*, see [Copying Variable Values from Contributing Events to the Rule Output Event](#) (see page 91).

Counter Event Threshold <N>

Sets a certain threshold that when reached, generates a new event. The rule will also generate another event when the count is lower than the threshold again.

Threshold violated event <V> - Event Code

Is generated when the count threshold is reached. To specify the event code, click *Browse*, select the event in the *Select Event* dialog (which displays all of the events loaded into *Event Configuration*), and click *OK*.

Threshold violated event <V> - Copy Event Variables

If you want to copy the values of any event variables in the first event that activated the rule to generated event <V>, select Copy Event Variables, and do one of the following:

- Select Copy All to copy the values of all of the variables into Threshold violated event <V>.
- Select Copy, and specify a comma-separated list of specific variable IDs whose values should be copied. Ranges of IDs are also supported, for example, "1-4".

Threshold reset event <R> - Event Code

Generates when the count is below the threshold again. To specify the event code, click Browse, select the event in the Select Event dialog (which displays all of the events loaded into Event Configuration), and click OK.

Threshold reset event <R> - Copy Event Variables

If you want to copy the values of any event variables in the first event that activated the rule to generated event Reason:, select Copy Event Variables, and do one of the following:

- Select Copy All to copy the values of all of the variables into Threshold reset event Reason:.
- Select Copy, and specify a comma-separated list of specific variable IDs whose values should be copied. Ranges of IDs are also supported, for example, "1-4".

Configuring Heartbeat Rule Settings

You configure heartbeat rule using the following dialog.

EC: Heartbeat event - SPECTRUM OneClick

Check for periodic heartbeat event <H>, which has to be seen at least every <P> seconds. When one is missing, generate event <Z>. The rule is instantiated through event 0x10002, and can be stopped with event <S>.

Event Variable Discriminators:

Heartbeat Event <H>

Event Code: Browse...

Copy Event Variables

Copy All

Copy of Event to

Heartbeat Period <P>

Heartbeat Period: Days +

Generate event on missing heartbeat <Z>

Event Code: Browse...

Copy Event Variables

Copy All

Copy of Event 0x10002 to

Optional: Heartbeat Rule Stop Event <S>

Event Code: Browse...

OK Cancel

The settings in the dialog include the following:

Event Variable Discriminators

A comma-separated list of the IDs of the event variables in the event to evaluate as a part of the rule.

Note: You must enter each ID; ranges of IDs are not supported. Event variable discriminators are a general feature available for all types of rules.

Heartbeat Event <H> - Event Code

Sets the Heartbeat Event code. To specify the event code, click Browse, select the event in the Select Event dialog (which displays all of the events loaded into Event Configuration), and click OK.

Heartbeat Event <H> - Copy Event Variables

If you want to copy the values of any event variables in the first event that activated the rule to generated event <H>, select Copy Event Variables, and do one of the following:

- Select Copy All to copy the values of all of the variables into Heartbeat Event <H>.
- Select Copy, and specify a comma-separated list of specific variable IDs whose values should be copied. Ranges of IDs are also supported, for example, "1-4".

Note: For information on the proper syntax to use when specifying variable IDs or ranges of IDs using Event Configuration, see [Copying Variable Values from Contributing Events to the Rule Output Event](#) (see page 91).

Heartbeat Period <P>

Sets the time gap between individual heartbeats.

Generate event on missing heartbeat <Z> - Event Code

Generates an event when the heartbeat event is missed. To specify the event code, click Browse, select the event in the Select Event dialog (which displays all of the events loaded into Event Configuration), and click OK.

Generate event on missing heartbeat <Z> - Copy Event Variables

If you want to copy the values of any event variables in the first event that activated the rule to generated event <Z>, select Copy Event Variables, and do one of the following:

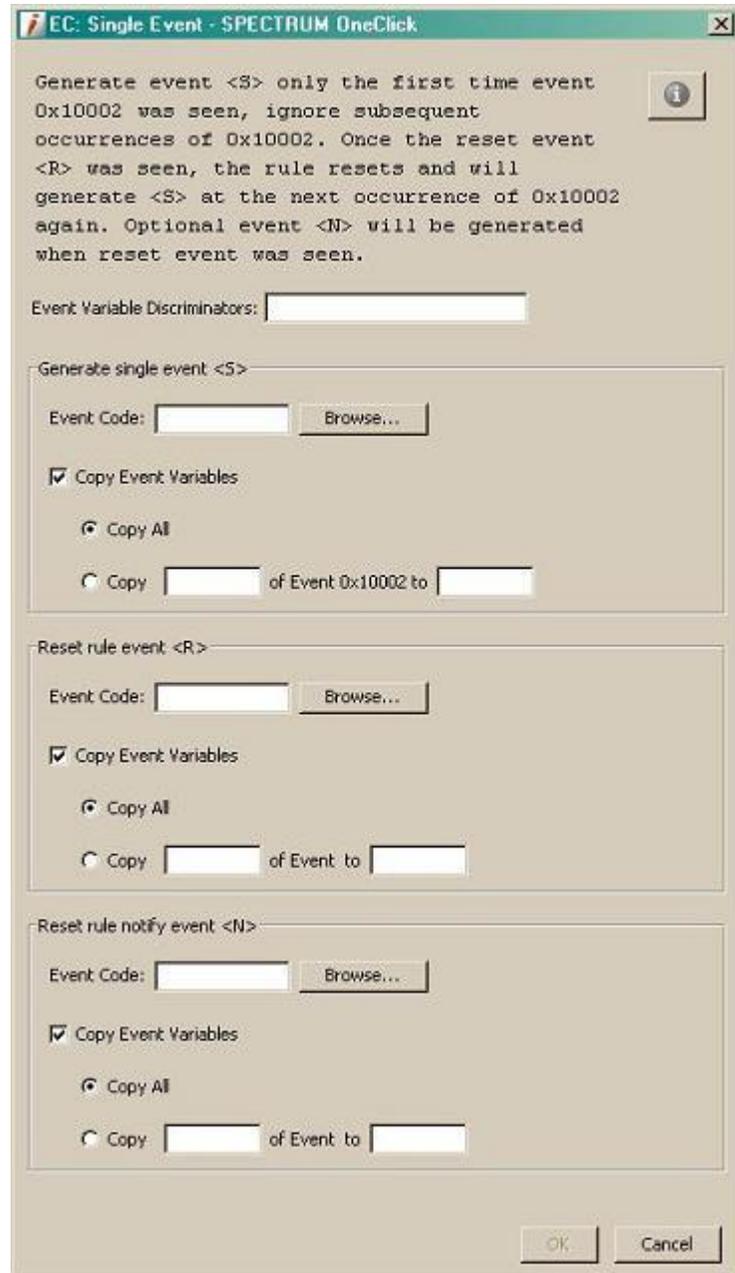
- Select Copy All to copy the values of all of the variables to Generate event on missing heartbeat <Z>.
- Select Copy, and specify a comma-separated list of specific variable IDs whose values should be copied. Ranges of IDs are also supported, for example, "1-4".

(Optional) Heartbeat Rule Stop Event <S> - Event Code

The Heartbeat Rule Stop Event stops the Heartbeat event rule. To specify the event code, click Browse, select the event in the Select Event dialog (which displays all of the events loaded into Event Configuration), and click OK.

Configuring Single Event Rule Settings

You configure a single event rule using the following dialog.



The settings in the dialog include the following:

Event Variable Discriminators

A comma-separated list of the IDs of the event variables in the event to evaluate as a part of the rule.

Note: You must enter each ID; ranges of IDs are not supported. Event variable discriminators are a general feature available for all types of rules.

Generate single event <S> - Event Code

It is generated the first time the trigger event is seen either when the rule is instantiated, or the first time the trigger event occurs after the reset event is seen. To specify the event code, click Browse, select the event in the Select Event dialog (which displays all of the events loaded into Event Configuration), and click OK.

Generate single event <S> - Copy Event Variables

If you want to copy the values of any event variables in the first event that activated the rule to generated event <D>, select Copy Event Variables, and do one of the following:

- Select Copy All to copy the values of all of the variables into Count down event <D>.
- Select Copy, and specify a comma-separated list of specific variable IDs whose values should be copied. Ranges of IDs are also supported, for example, "1-4".

Note: For information on the proper syntax to use when specifying variable IDs or ranges of IDs using Event Configuration, see [Copying Variable Values from Contributing Events to the Rule Output Event](#) (see page 91).

Reset rule event <R> - Event Code

Sets the reset event. To specify the event code, click Browse, select the event in the Select Event dialog (which displays all of the events loaded into Event Configuration), and click OK.

Reset rule event <R> - Copy Event Variables

If you want to copy the values of any event variables in the first event that activated the rule to generated event <D>, select Copy Event Variables, and do one of the following:

- Select Copy All to copy the values of all of the variables into Count down event <D>.
- Select Copy, and specify a comma-separated list of specific variable IDs whose values should be copied. Ranges of IDs are also supported, for example, "1-4".

(Optional) Reset rule notify event <N> - Event Code

It is generated when the reset event is seen. To specify the event code, click Browse, select the event in the Select Event dialog (which displays all of the events loaded into Event Configuration), and click OK.

Reset rule notify event <N>- Copy Event Variables

If you want to copy the values of any event variables in the first event that activated the rule to generated event <D>, select Copy Event Variables, and do one of the following:

- Select Copy All to copy the values of all of the variables into Count down event <D>.
- Select Copy, and specify a comma-separated list of specific variable IDs whose values should be copied. Ranges of IDs are also supported, for example, "1-4".

Configuring Solo Event Rule Settings

You configure a solo event rule using the following dialog.

Generate event <Z> when solo event <S> was seen, but only if none of the events in list <P> were seen within seconds before or <A> seconds after the solo event. The rule is instantiated through event 0x10002, and can be stopped with event <X>.

Event Variable Discriminators:

Prevent period before Solo Event

Time Interval: Days +

Solo Event <S>

Event Code:

Copy Event Variables

Copy All

Copy of Event to

Prevent period after Solo Event <A>

Time Interval: Days +

Generated Event <Z>

Generate Event:

Copy Event Variables

Copy All

Copy of Event 0x10002 to

Prevent Events <P>

Event Code	Copy From	Copy To

Stop Event <X>

Event Code:

The settings in the dialog include the following:

Event Variable Discriminators

A comma-separated list of the IDs of the event variables in the event to evaluate as a part of the rule.

Note: You must enter each ID; ranges of IDs are not supported. Event variable discriminators are a general feature available for all types of rules.

**Prevent period before Solo Event **

Sets the time period before the solo event where none of the 'prevent' events may occur (in seconds).

Solo Event <S>- Event Code

Specifies the Solo event. To specify the event code, click Browse, select the event in the Select Event dialog (which displays all of the events loaded into Event Configuration), and click OK.

Solo Event <S>- Copy Event Variables

If you want to copy the values of any event variables in the first event that activated the rule to generated event <D>, select Copy Event Variables, and do one of the following:

- Select Copy All to copy the values of all of the variables into Count down event <D>.
- Select Copy, and specify a comma-separated list of specific variable IDs whose values should be copied. Ranges of IDs are also supported, for example, "1-4".

Note: For information on the proper syntax to use when specifying variable IDs or ranges of IDs using Event Configuration, see [Copying Variable Values from Contributing Events to the Rule Output Event](#) (see page 91)

Prevent period after Solo Event Action:

Sets the time period after the solo event where none of the 'prevent' events may occur.

Generated Event <Z> - Event Code

Defines the event that will be generated when the rule triggers (when just the 'solo' event is seen). To specify the event code, click Browse, select the event in the Select Event dialog (which displays all of the events loaded into Event Configuration), and click OK.

Generated Event <Z> - Copy Event Variables

If you want to copy the values of any event variables in the first event that activated the rule to generated event <D>, select Copy Event Variables, and do one of the following:

- Select Copy All to copy the values of all of the variables into Count down event <D>.
- Select Copy, and specify a comma-separated list of specific variable IDs whose values should be copied. Ranges of IDs are also supported, for example, "1-4"

Prevent Events <P>

Defines a list of 'prevent' events. In the dialog, use the buttons above the list of prevent events to add new events and to modify, copy, and delete a selected event.

Stop Event <X> - Event Code

The Stop Event stops the Solo Event rule. To specify the event code, click Browse, select the event in the Select Event dialog which displays all of the events loaded into Event Configuration, and click OK.

Copy Variable Values from Contributing Events to the Rule Output Event

When you use event rules to trigger events, depending on the rule, the event that is generated as the output of the rule might be generated only after multiple, contributing events occur or certain complex conditions are met.

By default, a rule output event does not have any event variables. However, you can specify that the values of the event variables in the events that contribute to the rule's processing should be copied to the rule output event. This lets you specify event processing behaviors for the rule output event based on those values, which you can do using event variable discriminators. Moreover, because the values of event variables in events that generate alarms are also stored in those alarms, you can also use event discriminators to specify alarm processing (generation or clearing of alarms) based on the values. For an example of this usage, see [Event Variable Copying and Event Discriminators](#) (see page 157).

You can specify that all of the event variables in a contributing event should be copied to the rule output event, or you can specify one or more specific variables by their IDs. These are the event variable IDs that are mapped (in the AlertMap file) to the OIDs of the variable bindings sent with the trap.

Note: For more information about AlertMap files, see the *Event Configuration User Guide*.

To copy only specific variable binding values from a contributing event to the rule output event, enter a comma-separated list of the IDs and/or ranges of IDs, as shown in the following image.



Use the first text box to specify the event variables in the contributing event that you want to copy, and use the second text box to specify the event variables in the output event into which the values should be copied. The first ID in the "source" text box will be copied into the first ID in the "target" text box, and so on.

You can copy the value of one source variable into a target variable that has a different ID. For example, the preceding image indicates that source variable 1 should be copied into target variable 1, source variable 2 should be copied into target variable 3, and source variables 3, 4, and 5 should be copied into target variables 4, 5, and 6, respectively.

Modifying Event Rules

You can modify event rules.

To modify event rules

1. In the Navigation panel, select the event that activates (triggers) the event rule.
2. In the Details panel, click the Event Rules tab.

The list of event rules for the selected event appears.

3. Select the event rule to modify, and click  (edit).
4. Modify the configuration of the event rule as required.
5. Click OK.

More information:

- [Configuring Event Condition Rule Settings](#) (see page 66)
- [Configuring Event Pair Rule Settings](#) (see page 70)
- [Configuring Event Rate Rule Settings](#) (see page 73)
- [Configuring Event Series Rule Settings](#) (see page 76)

Delete Event Rules

You can delete event rules.

To delete event rules

1. In the Navigation panel, select the event that activates (triggers) the event rule.
2. In the Details panel, click the Event Rules tab.

The list of event rules for the selected event appears.

3. Select the event rule to delete, and click  (delete).
4. Click OK to confirm the deletion.

Chapter 5: Using Procedures in Event Processing

Using Procedures in Event Disposition Files

CA Spectrum includes many procedures that accomplish common tasks. These are located in `<$SPECROOT>/SS/CsVendor/CA/Procedures`. Each procedure XML file defines the procedure (input parameters, return value, and so on) and provides documentation about its function.

You can specify that one or more procedures should be executed when an event is processed by adding the procedures to the event's event map in the appropriate event disposition file. When the event is generated for a given model, the SpectroSERVER processes the event and executes the procedures. (Both the current event and the current model are required for proper procedure execution, and both are automatically set by the SpectroSERVER.)

To add a procedure to an event map, use the following syntax:

```
<event_code> E <event_severity> <processing_parameters> P <procedure>
```

<event_code> S

Specifies the event code of the event.

E

Identifies the action code that indicates the event data should be logged by the Archive Manager in the Distribute Data Manager (DDM) database.

<event_severity>

Specifies the relative severity of the event on a scale of 0 to 100.

<processing_parameters>

Specifies additional event processing behaviors, such as whether the event generates an alarm, clears one or more alarms, or executes one or more event rules.

P

Specifies the action code that indicates that a procedure follows.

<procedure>

Specifies the procedure to execute. To specify multiple procedures, use multiple instances of `P <procedure>`.

Example:

The following event map executes a procedure that creates event 0xabcd0002 on a port's device model, which is retrieved by reading the Device_Mdl_Handle attribute (0x10069). Event 0xabcd0002 has copies of all of the event variables in the original event.

```
0xabcd0000 E 50                                \  
  P "CreateEventWithVariables(                  \  
    ReadAttribute( { C CURRENT_MODEL }, { H 0x10069 } ), \  
    { H 0xabcd0002 },                            \  
    GetEventVariableList()                      \  
  )"                                             \  
"
```

In the example, CURRENT_MODEL is a predefined constant referring to the current model, which is automatically set in the environment by the SpectroSERVER when the event is processed.

As shown in the example, a procedure must be fully enclosed in double quotes, has a procedure name, and takes parameters. Parameters follow the procedure name, are enclosed in braces if they are direct values (or constants), and are separated by commas.

If a procedure spans multiple lines, you must use the line continuation character to indicate that each line continues a single event map.

You can use any number of procedures in an event map.

Note: The user-defined event procedures are read-only in the Event-Configuration editor.

Input Parameters

A procedure accepts the following types of input parameters:

- A value. Values are specified by type and value.
- A constant. Constants are specified by the letter C followed by the reserved word for the constant.
- Another procedure, which is evaluated when the parameter is needed and whose return value is used as the actual input parameter for the calling procedure.

Parameter Types

When you specify a direct value as a parameter, you must specify its type. Value types have a derivation hierarchy, which is shown in the following table. You can use a more specialized type for a more general type. For example, you can use a model handle in place of an unsigned integer, and you can use either of these types in place of a number.

When a parameter has a type that is not the same as or a derivative of the expected type, if the type is similar, most procedures attempt a conversion to the expected type. If a conversion fails or is not performed, an error occurs.

Note: You can use the ToUInteger procedure (provided with CA Spectrum) to convert a parameter value to an unsigned integer. Many times, this can help to avoid errors due to expected versus actual parameter types.

To specify a parameter type in a procedure, you specify a letter (a short symbol) for the type. For example, both of the parameters in the following procedure are handles (IDs), which are indicated by the letter 'H.' The first handle specifies the model for which to generate the event, and the second handle specifies the event to generate.

```
CreateEvent( { H 0x29c00003 }, { H 0xffff0000 } )
```

The table that follows provides a short list of parameter types, their derivation hierarchy, and their associated short symbols. For the complete list of short symbols, see [EventCondition Rule](#) (see page 135).

Parameter Type	Description	Short Symbol
Object	Derivation root	N/A
List	List root	N/A
AttributeList	List of attribute values	N/A
EventVariableList	List of event variables	N/A
Attribute Value	Any possible attribute value	N/A
Number	Number root	N/A
Integer	An integer value	I
Unsigned Integer	An unsigned integer value	U
DateTime	A time value	T
ModelHandle	A model handle	H
ModelType	A model type handle	H

Parameter Type	Description	Short Symbol
RelationHandle	A relation handle	H
Attribute ID	An attribute ID	a
Unsigned Long	An unsigned long (64-bit) value	L
Boolean	A Boolean value	B
Double	A real value	R
String	String root	N/A
Text String	A String value	S
Octet String	An octet String value	O
Tagged Octet String	A hexadecimal tagged octet String	X
Object ID	An object ID value	o
IP Address	An IP address	A
Variable	The value of a variable	V

Constants

You specify a constant using the letter C followed by the reserved word for the constant. For example, the following procedure creates event 0xffff0000 for the current model:

```
CreateEvent( {C CURRENT_MODEL}, { H 0xffff0000 } )
```

You can use the following list of constants in procedures in event maps:

CURRENT_EVENT

The current event, which is set in the environment automatically by the SpectroSERVER.

CURRENT_MODEL

The current model, which is set in the environment automatically by the SpectroSERVER.

RELATION_SIDE_LEFT

The model on the left side of an association.

RELATION_SIDE_RIGHT

The model on the right side of an association.

RELATION_SIDE_EITHER

The model on either side of an association.

Return Values

Every procedure returns a value whose type depends on the function that the procedure performs, for example, a list of model handles or a Boolean value of TRUE or FALSE.

The possible types of return values are the same as the possible types of input parameters for procedures. This lets you use procedures as input parameters because they ultimately evaluate to permissible input values.

Examples of Procedures in Event Disposition Files

The following examples show how procedures can be used in event maps to implement event processing. Assume that each procedure is wrapped inside double quotes.

Add together two integer numbers (4 and 16):

```
Add( { I 16 }, { I 4 } )
```

Add together a direct value of 4 and the value in event variable 2 in the current event:

```
Add( { U 4 }, GetEventVariable( { U 2 } ) )
```

Create event 0xffff0000 for model 0x29c0003:

```
CreateEvent( { H 0x29c00003 }, { H 0xffff0000 } )
```

Create event 0xffff0000 if event variable 1 in the current event exceeds 100:

```
If( Less( { U 100 }, GetEventVariable( { U 1 } ) ),      \
    CreateEvent( { H 0x29c00003 }, { H 0xffff0000 } ),  \
    Nil()                                               \
)
```

Create event 0x10002 with a list of event variables. The list will be a copy of the variables in the current event. In the new event, also set event variable 2 to 1965:

```
CreateEventWithVariables( { H 0x29c00003 },           \
    { H 0x10002 },                                     \
    SetEventVariable( GetEventVariableList( { C CURRENT_EVENT } ), \
    { U 2 },                                           \
    { U 1965 }                                         \
    )                                                  \
)
```

Create a list and add several elements. The list will contain the following elements in this order: 5,3,2,1,4,6:

```
AddTail(                                \
  AddHead(                               \
    AddTail(                              \
      AddHead(                            \
        AddHead(                          \
          AddTail(                        \
            CreateList(),                 \
              { U 1 } ),                  \
              { U 2 } ),                  \
              { U 3 } ),                  \
              { U 4 } ),                  \
              { U 5 } ),                  \
            { U 6 } )
```

Create event 0x10002 for model 0x29c00003 if the model has IP address 191.168.102.25:

```
If ( HasIPAddress( { H 0x29c00003 }, { A 192.168.102.25 } ), \
  CreateEvent( { H 0x29c00003 }, { H 0x10002 } ),           \
  Nil()                                                       \
)
```

Create event 0xffff0000 for model 0x29c00003 if attribute 0xffff0000 of the current model has the same value as event variable 1 in the current event:

```
If( Equals( ReadAttribute( { C CURRENT_MODEL }, { H 0xffff0000 } ), \
  GetEventVariable( { U 1 } ) )                                     \
),                                                                    \
  CreateEvent( { H 0x29c00003 }, { H 0xffff0000 } ),              \
  Nil()                                                             \
)
```

Return TRUE if the current time is 8 a.m. or later:

```
GreaterOrEqual( GetHour( GetCurrentTime() ), { U 8 } )
```

Retrieve the value of event variable 1 in the current event and write it to attribute 0xffff0001 in the current model:

```
WriteAttribute( { C CURRENT_MODEL }, { H 0xffff0001 }, GetEventVariable( { U 1 } )
)
```

Create event 0xffff0000 for model 0x29c00003 if exactly 3 event variables in the current event's event variable list have a value of 4. The procedure iterates over the list, assigning each element in the list to the loop variable 'X' in turn and checking its value. If the value of the element is 4, the return variable 'ret' is incremented. After iterating through the list, the return value is checked, and the event is generated if its value is 3:

```

    If( Equals( ForEach( GetEventVariableList(),           \
                      { Variable X },                   \
                      { Variable ret },                 \
                      { U 0 },                          \
                      If( Equals( { Variable X },       \
                                { U 4 } ),             \
                          Assign( { Variable ret },    \
                                Add( { Variable ret }, \
                                    { U 1 } )          \
                                ),                      \
                          Nil()                       \
                      )                                \
    ),                                               \
    { U 3 } ),                                       \
    CreateEvent( { H 0x29c00003 }, { H 0xffff0000 } ), \
    Nil()                                           \
)

```

Logging Errors in Event Procedures

You can log the errors that the SpectroSERVER encounters while parsing a procedure XML file or while executing a procedure used in an event map to an error file.

Errors that are found while parsing a procedure file are written to the file specified in the `procedure_error_file` parameter in the `.vnmrc` resource file for the SpectroSERVER. Errors that are found while executing a procedure used in an event map are written to the file specified in the `event_disp_error_file` parameter. For more information on these parameters, see [Logging Event-Related Errors](#) (see page 25).

More information:

[Logging Event-Related Errors](#) (see page 25)

Troubleshooting Event Procedures

You can use the DebugValue procedure to assist you in troubleshooting procedures used in event processing that are not performing as you intend. The DebugValue procedure prints the value of a parameter to the error log file specified in the event_disp_error_file parameter in the .vnmrc file for the SpectroSERVER.

Note: For information on how to set the event_disp_error_file parameter in the .vnmrc file, see [event_disp_error_file](#) (see page 25).

The DebugValue procedure accepts the following input parameters:

- A text string that is printed before the value is printed. This parameter is especially helpful for differentiating values when you are using more than one DebugValue procedure in a given event procedure.
- Any value (value, constant, or another procedure).

The DebugValue procedure returns the value that is passed in as the second input parameter (which is unchanged). This means that you can use it as an input parameter in any other procedure without affecting the execution of that procedure.

As an example, the following DebugValue procedure prints a text string followed by the value of a read operation on an attribute of the current model. It then returns the value of the read operation, which, in turn, is used as the first input parameter in the Equals procedure.

```
If( Equals( DebugValue( { S \"Attribute value is:\" },          \
                ReadAttribute( { C CURRENT_MODEL }, { H 1 } ),    \
                GetEventVariable( { U 1 } )                      \
            ),          \
        CreateEvent( { H 0x29c00003 }, { H 0xffff0000 } ),      \
        Nil()          \
    )
```

Appendix A: AlertMap Files

More information:

[SNMPv2 Support](#) (see page 111)

[AlertMap File Syntax](#) (see page 107)

[SNMP Trap Overview](#) (see page 103)

SNMP Trap Overview

An SNMP trap is sent out as a trap PDU (Protocol Data Unit). The PDU contains the following pieces of information:

Enterprise OID

Identifies the company responsible for a device sending the trap. For example, 1.3.6.1.4.1.X is an enterprise OID where X identifies the enterprise (for example, Sun). The numbers preceding the X represent a hierarchy of global bodies responsible for the management of information. The Internet Assigned Numbers Authority (IANA) allocates the enterprise level numbers that identify companies and their management MIBs on the MIB tree. For more information, see their website, <http://www.iana.org>.

Network Address

Specifies the network address of the managed element initiating the trap.

Generic Trap Identifier

This can be a value from 0 through 6. There are six standard industry traps within the SNMP protocol. These traps have a generic trap identifier of 0-5. The number 6 indicates that the trap is an enterprise-specific trap.

Enterprise specific traps are proprietary traps created for developer-specific types of managed nodes. These traps are defined by proprietary MIBs.

Specific Trap

Specifies the specific trap number as listed in the trap definition of the trap MIB.

Time Stamp

Specifies the time at which the trap was created.

Variable Bindings

Specifies the variables and values that are defined in the trap. These variables are generally pointers to other MIB objects.

For example, when a redundant power supply fails in a Cisco router or switch, the following information is sent in the trap PDU:

- Network address of the device
- Timestamp of the trap
- Enterprise OID: 1.3.6.1.4.1.9
- Generic Trap ID: 6
- Specific Trap ID: 5
- Variable Binding(s): 1.3.6.1.4.1.9.9.13.1.5.1.2,
1.3.6.1.4.1.9.9.13.1.5.1.3

The network address and the timestamp information vary depending on the device and the time that the trap was sent.

The enterprise OID identifies the vendor company using the last two digits. In this example, 9 indicates a trap generated by a Cisco device.

The generic trap ID of 6 indicates that this trap is an enterprise-specific trap defined by a proprietary MIB. The specific trap ID of 5 is the number assigned to the trap in the referenced Cisco MIB. The following portion of the Cisco environment monitoring MIB (CISCO-ENVMON-MIB) defines this trap.

```
ciscoEnvMonRedundantSupplyNotification TRAP-TYPE
-- Reverse mappable trap
    ENTERPRISE ciscoEnvMonMIBNotificationPrefix
    VARIABLES {
        ciscoEnvMonSupplyStatusDescr, ciscoEnvMonSupplyState }
-- Status
-- mandatory
    DESCRIPTION
        "A ciscoEnvMonRedundantSupplyNotification is sent if the

        redundant power supply (where extant) fails. Since such a
        notification is usually generated before the shutdown state is
        reached, it can convey more data and has a better chance of being
        sent than does the ciscoEnvMonShutdownNotification."
    ::= 5
```

This trap definition has two variables (ciscoEnvMonSupplyStatusDescr and ciscoEnvMonSupplyState) that are sent as variable bindings. Such variables are actually references to other managed objects defined by the MIB. The OIDs sent in the variable bindings represent these managed objects.

The following excerpts from the MIB define `ciscoEnvMonSupplyStatusDescr` and `ciscoEnvMonSupplyState`:

```

ciscoEnvMonSupplyStatusDescr OBJECT-TYPE
    SYNTAX DisplayString(SIZE(0..32))
    --          Rsyntax OCTET STRING(SIZE(0..32))
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "Textual description of the power supply being instrumented.
        This description is a short textual label, suitable as a human-
        sensible identification for the rest of the information in the
        entry."
    ::= { ciscoEnvMonSupplyStatusEntry 2 }
ciscoEnvMonSupplyState OBJECT-TYPE
    SYNTAX CiscoEnvMonState
    --          Rsyntax INTEGER {
    --              normal(1),
    --              warning(2),
    --              critical(3),
    --              shutdown(4),
    --              notPresent(5),
    --              notFunctioning(6)
    --          }
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The current state of the power supply being instrumented."
    ::= { ciscoEnvMonSupplyStatusEntry 3 }

```

The `ciscoEnvMonSupplyStatusDescr` variable will have a string value describing the power supply, and the `ciscoEnvMonSupplyState` variable can have a value of 1, 2, 3, 4, 5 or 6 depending on its status. The current value for each variable is detected using the variable bindings in the trap PDU.

About Mapping a Trap to a CA Spectrum Event

When CA Spectrum receives a trap, it must be associated with a specific CA Spectrum event for CA Spectrum to use the trap information. This association is made in an AlertMap file.

CA Spectrum can receive SNMP traps from devices that have been modeled using the Pingable model type. To enable this, you must set the `enable_traps_for_pingables` variable in the CA Spectrum `.vnmrc` file to TRUE. For information about how to do this, see the *Distributed SpectroSERVER Administrator Guide*. The AlertMap file for the Pingable model type is located in the `<$SPECROOT>\SS\CsVendor\Cabletron\Pingable` directory.

About Processing Alerts with AlertMap Files

The IP address of the managed element issuing the trap is sent as a part of the trap information. CA Spectrum uses the IP address to identify the model that represents the managed element issuing the trap. From this information, CA Spectrum is able to identify the model type associated with the managed element.

To process the trap correctly, CA Spectrum refers to the AlertMap file. Depending on where they reside, AlertMap files can be applied globally (that is, to all CA Spectrum model types) or only to models of a specific model type.

Thus, if you want to change the trap mapping for a particular model type only, you should modify the AlertMap file for that model type. If an AlertMap file at the model type level does not exist, you must create one and make your changes there.

If you want to change how an alert is mapped globally (that is, for all model types), you will need to find all AlertMap files that specify a mapping for that trap, and make your changes to each one of these files.

Note: Global AlertMap files apply to all vendor model types, not just those created by the vendor who created the file.

AlertMap File Location

Default trap mappings are located in the `<$SPECROOT>/SS/CsVendor` directory while MIB Tools customized trap mappings are located in the `<$SPECROOT>/custom/Events/AlertMap` directory. MIB Tools' customized trap mappings are preserved when you upgrade CA Spectrum, and they take precedence over any existing default trap mappings. For more information about working with custom trap mappings and default trap mappings, see Map Tab.

Global AlertMap files are located in `<$SPECROOT>/SS/CsVendor/<developername>/AlertMap`, where `<developername>` is usually the name of the vendor, manufacturer, or developer associated with the model type.

In some instances, CA may use something other than the developer name for the directory. For example, the IETF directory contains the AlertMap file that maps standard RFC traps.

If the AlertMap file is specific to the model type, it is located in
`<$SPECROOT>/SS/CSVendor/<developername>/
<model_type_name>AlertMap`, where:

`<developername>` is the name of the vendor, manufacturer, or developer associated with the model type (for example, Compaq). If you are developing your own management module, your developer name is used here.

`<model_type_name>` is the name of the model type.

If a mapping for a trap exists in both a model type AlertMap file and a global AlertMap file, the model type AlertMap file takes precedence for that particular model type.

When you map traps using MIB Tools, entries are generated in the following file on all SpectroSERVERs in your DSS environment:

`<$SPECROOT>/custom/Events/AlertMap`

The mapping information for a trap in these files overrides any mapping information that previously existed for that same trap in other files or directories on the SpectroSERVER. For more information, see Chapter 10: Managing MIBs and Traps.

The name of every AlertMap file is always "AlertMap." No file extension is used in the Windows environment.

AlertMap File Syntax

If the appropriate AlertMap file is found, CA Spectrum looks for an entry matching the trap. Each entry in the Alert Map file has three components: the alert code, the event code, and the OID map.

1.3.6.1.4.1.9.6.5 0x180000 1.3.6.1.4.1.9.9.13.1.5.1.3(1,0)

▲
▲
▲
Alert Code Event Code OID Map

Alert Code

The alert code consists of three pieces of information from the trap: the Enterprise OID string, the generic trap identifier, and the specific trap identifier.

1.3.6.1.4.1.9: Enterprise OID (in this case indicating a Cisco device).

6: The Generic Trap Identifier (in this case the 6 indicates an Enterprise specific trap).

5: Specific Trap Identifier (in this case 5).

Event Code

The event code is a 4-byte integer expressed in hexadecimal format. Each event code has two parts: the first two bytes contain the developer ID of the developer who created the file, and the last two bytes identify the event with a unique number relative to all other event codes for that particular developer.

If the event code is zero, an event will not be generated for that particular alert.

If the event code is non-zero, CA Spectrum will map the alert variables to event variables using the OID map.

OID Map

The trap you are mapping may include one or more pieces of variable information, known as variable bindings. Each of these variable bindings provides information about the trap. You can choose to map these bindings to event variables so that the information can be used by CA Spectrum.

Variable bindings and the information about how CA Spectrum should map them are specified in the OID map section of the alert map entry. A single alert map entry may have multiple OID maps associated with it indicating that there were many variable bindings sent with the SNMP trap. These OID maps should be separated by the backslash (\) character and a new line as in the following example.

```
1.3.6.1.4.1.9.6.5 0x180000 1.3.6.1.4.1.9.9.13.1.5.1.3(1,2) \  
1.3.6.1.4.1.9.9.13.1.5.1.2(4,0)
```

As shown in the following example, the OID map can be broken down into three parts: the OID, the value variable ID, and the instance variable ID.



The OID identifies the specific variable being sent with the trap. The previous OID references the `ciscoEnvMonSupplyState` variable in the Cisco Environment monitoring MIB (`CISCO-ENVMON-MIB`).

The value variable ID stores the value of the variable sent in the variable binding. Any integer value can be used here; however, it must be different from the integer used for the instance variable ID. A value of zero indicates that you do not want to store the value of the variable binding.

The instance variable ID stores the instance portion of the OID. If your variable binding identifies a particular object from a table variable within the trap MIB, it will likely include an instance ID. Any integer value can be used here; however, it must be different from the integer used for the value variable ID. A value of zero indicates that you do not want to store the value of the instance variable.

Important! There are no spaces between the OID, the value variable ID, and the instance variable ID.

Comments

There are two comment identifiers that allow you to add comments to the AlertMap file: `#` or `//`. Text entered after one of these identifiers and before the start of the next line is ignored when processing the AlertMap file.

For example:

```
#Comment
0.0 0x10306 #Comment
1.0 0x10307 //Comment
2.0 0x220001 1.3.6.1.2.1.2.2.1.1(1,2)
3.0 0x220002 1.3.6.1.2.1.2.2.1.1(1,2)\
              1.3.6.1.2.1.2.2.1.3(4,0)
//Comment
//Comment
4.0 0x1030a
5.0 0x1030b
1.3.6.1.4.1.45.6.271 0x1060f 1.3.6.1.4.1.45.1.2.1.9.2.1.2(3,4) #Comment
```

As shown in [OID Map](#) (see page 108), the backslash (`\`) character is used as a line continuation character. You can also use comments when using the line continuation character.

For example:

```
3.0 0x220002 1.3.6.1.2.1.2.2.1.1(1,2)\ #Comment
            1.3.6.1.2.1.2.2.1.2(3,0)\
                //Comment
            1.3.6.1.2.1.2.2.1.3(4,0)
```

Note: Inline and multiline comments are not supported.

How CA Spectrum Maps Alert Variables to Event Variables

CA Spectrum maps alert variables to event variables as follows:

1. CA Spectrum scans the AlertMap file entry to find an OID Map whose OID either exactly matches the trap variable's OID, or is a prefix of the trap variable's OID.
2. If more than one OID map has an OID that is a prefix of the trap variable's OID, CA Spectrum chooses the OID with the longest prefix (best match).
3. If an OID map is found, CA Spectrum examines its value variable ID to decide whether to translate the trap variable's value into an event variable. If an OID map is not found, CA Spectrum ignores the alert variable.
4. If the value variable ID is zero, CA Spectrum ignores the trap variable's value. Otherwise, the type, length, and value of the event variable are obtained from the trap variable, and an event variable is constructed. The value variable ID can then be used to represent the value of the variable binding in an Event Format file.
5. CA Spectrum examines the OID Map to decide whether to translate the trap variable's instance ID into an event variable. If the OID in the OID Map exactly matches the trap variable's OID, there is no instance ID to translate. If the instance variable ID in the parameter is zero, the instance OID is ignored. If the instance ID exists and the instance variable ID is non-zero, an event variable is constructed. The instance variable ID can then be used to represent the value of the OID instance in an Event Format file.

More information:

[Event Format Files](#) (see page 159)

Error Messages

If CA Spectrum is unable to identify the model for a given IP address, an event is generated on behalf of the VNM model indicating that a trap was received from an unknown SNMP device (event 0x00010802). Contained within this event are details about the trap, including the agent IP address, enterprise OID, trap code, community name, and variable binding data.

If CA Spectrum can properly identify the model for the trap source, but it cannot find an entry for the specific trap code in the AlertMap file, an event indicating that an unknown alert was received (event 0x00010801) is generated on behalf of that model. Contained within the event are details about the trap, including the agent IP address, device type, device time, trap type, and variable binding data.

SNMPv2 Support

CA Spectrum supports the receipt and processing of SNMPv2 format InformRequests and traps as defined by RFC 2576, "Coexistence between Version 1, Version 2, and Version 3 of the Internet-Standard Network Management Framework."

InformRequest Support

When CA Spectrum receives an SNMPv2 InformRequest, it is decoded and converted to SNMPv1 format as specified in RFC 2576. A response to the InformRequest is generated and sent back to the inform originator.

How an SNMPv2 Trap is Mapped to a CA Spectrum Event

If you are mapping SNMPv2 traps to CA Spectrum events, you will need to reference the translated trap in the AlertMap file.

The following is an overview of how an SNMPv2 trap is translated. For complete information about this process, see RFC 2576. Section 3.2 of the RFC explains how an SNMPv2 trap is translated into an SNMPv1 trap so that it can be used in an SNMPv1 environment.

1. The SNMPv1 enterprise OID for the trap is determined as follows:
 - a. If the SNMPv2 snmpTrapOID parameter is one of the standard traps (defined in RFC 1907), then set the value of the SNMPv1 enterprise parameter to the value of the snmpTrapEnterprise.0 variable binding if it exists. If it does not exist, set the value to snmpTraps.
 - b. If the SNMPv2 snmpTrapOID parameter is not a standard trap, the following criteria is used:

If the next-to-last sub-identifier of the snmpTrapOID is 0, then find the enterprise number by removing the last two sub-identifiers.

snmpTrapOID = 1.3.6.1.4.1.1916.4.1.0.1

Enterprise number = 1.3.6.1.4.1.1916.4.1

If the next-to-last number is not 0, then find the enterprise number by removing the last sub-identifier.

snmpTrapOID = 1.3.6.1.4.1.5486.1.3.8

Enterprise number = 1.3.6.1.4.1.5486.1.3
2. The SNMPv1 generic trap identifier for the trap is determined as follows:
 - a. If the SNMPv2 snmpTrapOID parameter is a standard trap as defined in RFC 1907, use the standard generic trap parameter for that trap (0-5).

standard trap = 1.3.6.1.6.3.1.1.5.1 (cold start)

generic trap identifier = 0
 - b. If the SNMPv2 snmpTrapOID parameter is not a standard trap, set the generic-trap parameter to 6.

snmpTrapOID = 1.3.6.1.4.1.5486.1.3.8

generic trap identifier = 6
3. The SNMPv1 specific trap parameter is determined as follows:
 - a. If the SNMPv2 snmpTrapOID parameter is a standard trap as defined in RFC 1907, set the specific-trap parameter to 0.

standard trap = 1.3.6.1.6.3.1.1.5.1 (cold start)

specific trap parameter = 0
 - b. If the SNMPv2 snmpTrapOID parameter is not a standard trap, set the specific-trap parameter to the last sub-identifier of the SNMPv2 snmpTrapOID parameter.

snmpTrapOID = 1.3.6.1.4.1.5486.1.3.8

specific trap parameter = 8
4. The SNMPv2 variable bindings are converted directly to SNMPv1 bindings. As noted in RFC 2576, variable bindings of type Counter64 cannot be translated.

Example:

If you are sending the following SNMPv2 snmpTrapOID to the SpectroSERVER:
1.3.6.1.4.1.5486.1.3.8

You should use the following OID as the [alert code](#) (see page 107) in the AlertMap file: 1.3.6.1.4.1.5486.1.3.6.8.

Enterprise OID: 1.3.6.1.4.1.5486.1.3

Generic Trap Number: 6

Specific Trap Number: 8

Appendix B: Event Disposition Files

This section contains the following topics:

[About Event Disposition Files](#) (see page 115)

[Location of Event Disposition Files](#) (see page 116)

[File Syntax of Event Disposition Files](#) (see page 118)

[Generating Alarms](#) (see page 119)

[Clearing Alarms](#) (see page 124)

[Clearing Alarms Regardless of Event Discriminator Values](#) (see page 128)

[About Defining Event Rules](#) (see page 129)

[Syntax Errors in EventDisp Files](#) (see page 157)

[Add Comments in EventDisp Files](#) (see page 158)

About Event Disposition Files

When an SNMP trap is received by CA Spectrum and mapped to a CA Spectrum event that is generated for the model, an *event disposition file* is used to determine how to process the event. The processing instructions in an event disposition file (an ASCII text file) can include:

- Whether the event should be logged
- The severity of the event
- Whether the event should generate an alarm of a specific severity
- Whether the event should clear one or more alarms
- Whether the event should trigger an event rule (a series of events that are watched for and trigger another event if they occur in a specific pattern or time frame)

The events provided with CA Spectrum all have processing instructions defined for them in global and model type-specific event disposition files. In addition, whenever you create a new, custom event (either using MIB Tools when you are mapping a trap to the new event or later using Event Configuration) a custom event disposition file is automatically created. This means that typically you should not need to manually create an event disposition file.

Important! It is recommended that you specify the processing instructions for all events using MIB Tools and Event Configuration, both of which can save your customizations to one or more landscapes. If you are adding new trap support for a new device, first use MIB Tools to map the traps to new CA Spectrum events and specify basic event settings. Then, you can launch Event Configuration (directly from MIB Tools) and completely configure the events. Using this workflow, typically you should not need to manually modify an event disposition file; however, this appendix provides reference information on the proper syntax to use if manual modifications are ever required.

More information:

[Location of Event Disposition Files](#) (see page 116)

[File Syntax of Event Disposition Files](#) (see page 118)

Location of Event Disposition Files

The name of every event disposition (EventDisp) file is always EventDisp.

Note: In the Windows environment, no file extension is used.

The EventDisp files that support the events provided with CA Spectrum are installed in the following folders:

- <\$SPECROOT>/SS/CsVendor/<developer_name>

An EventDisp file in this location defines the processing for all events created by the developer that are *global* in scope. <developer_name> is the name of the developer, vendor, or manufacturer who created the EventDisp file.

Note: In some instances, CA may use something other than the developer name for the developer-specific directory; for example, the IETF directory contains the EventDisp file that specifies the processing for events resulting from standard RFC traps.

- <\$SPECROOT>/SS/CSVendor/<developer_name>/<model_type_name>

An EventDisp file in this location defines the processing for all events created by the developer and whose scope is *limited to the model type* represented by <model_type_name>. In this case, <developer_name> is the name of the developer, vendor, or manufacture who created the model type (for example, Compaq).

Note: If an event map entry exists in both a global EventDisp file and a model type-specific EventDisp file, the event map entry in the model type-specific file takes precedence.

If you customize the event processing for the CA-authored events provided with CA Spectrum, or if you create new, custom events using MIB Tools or Event Configuration, a custom EventDisp file is created in the following folder or in one of its subfolders:

<\$SPECROOT>/custom/Events

Note: The event map entries in a custom EventDisp file override those in the EventDisp files provided by default with CA Spectrum.

File Syntax of Event Disposition Files

Each line in an event disposition (EventDisp) file is called an *event map*, and each event map can specify one or more event processing behaviors, such as whether the event should be logged or whether it should generate an event. The following syntax is used for an event map:

```
<eventcode> E <eventseverity> <processing parameters> S <sbgw processing flag>
```

<eventcode>

Specifies the event code of the CA Spectrum event for which the processing behavior is being defined. The event code is defined in the AlertMap file that maps the trap to the event, or it is specified within the code that generates the event. The event code must be specified for CA Spectrum to process the event.

E

(Optional) Indicates that the event data should be logged by the Archive Manager to the Distributed Data Manager (DDM) database. If the E flag is not used, the event data is temporarily logged, but it is not preserved when the SpectroSERVER is shut down and then restarted.

<eventseverity>

Specifies the relative severity of the event on a scale of 0 to 100, where 0 is the least severe and 100 is the most severe. If a value for event severity is not specified, CA Spectrum uses a default value of 0.

Note: The event severity parameter is not currently used by the event management system. However, if you are logging the event, you are advised to assign an event severity value since use of this parameter may be incorporated into event processing in the future. If you are not logging the event, an event severity value should not be assigned.

<processing parameters>

Specifies additional event processing behaviors, such as whether the event generates an alarm or clears one or more alarms. The topics in this section provide detailed information on the proper syntax for various event processing behaviors.

S <sbgw processing flag>

(Optional) Specifies whether an event should be registered for Southbound Gateway processing or not. It applies to modeltype specific entries only and only those modeltypes derived from the southbound modeltype fragment. The S is followed by either of the following sbgw processing flags:

- + (process for Southbound Gateway: S+)
- - (do not process for Southbound Gateway: S-)

Because the default sbgw processing flag setting is '+', an entry without the 'S' flag present will always be processed by the Southbound Gateway. Setting any entry to 'S-' adds a modeltype specific EventDisp action without it being processed by the Southbound Gateway.

While an event does not require an event map in an EventDisp file, most events have one. If an event does *not* have an event map in an EventDisp, the event is logged in the DDM database by default (which means it is preserved when the SpectroSERVER is shut down and then restarted), but no additional processing takes place. If an event *does* have an event map, the event is processed according to the event map.

Note the following about EventDisp files:

- Empty lines in the EventDisp file are ignored.
- If a single event map spans more than one line, a backslash '\' at the end of each line must be used to help ensure that CA Spectrum considers the next line as a part of the event map.

Generating Alarms

The following syntax is used to generate an alarm:

```
<eventcode> E <eventseverity> A <alarmseverity>,<alarmcause>
```

A

Indicates that an alarm should be generated when the specified event occurs.

<alarmseverity>

Specifies a number between 0 and 6 that identifies the severity of the alarm:

- 0 (Normal)
- 1 (Minor)
- 2 (Major)
- 3 (Critical)
- 4 (Maintenance)
- 5 (Suppressed)
- 6 (Initial)

Each severity in the preceding list is associated with a color-coded condition. When an alarm is asserted on a model, the associated condition color is displayed on the model's icon to reflect the alarm status.

You can also specify an alarm severity of Variable or Conditional, each of which evaluates to one of the numeric severity levels. The following syntax is used to indicate a severity level of Variable:

```
{ v <event_variable_ID> }
```

<event_variable_ID>

Specifies the ID of the event variable in the event to evaluate to determine the alarm severity level.

Similarly, the following syntax is used to indicate a severity level of Conditional:

```
{ v <event_variable_ID> <folder_name>.<file_name> }
```

<event_variable_ID>

Specifies the ID of the event variable in the event to evaluate.

<folder_name>

Specifies the severity mapping file to use to determine the alarm severity level.

<file_name>

Specifies the severity mapping file to use to determine the alarm severity level.

Note: If a severity is not specified in the event map, CA Spectrum uses a default value of 0 (Normal).

<alarmcause>

Specifies a number that is used to identify the probable cause file that contains the messages associated with the alarm.

Example:

The following event map serves as an example:

```
0x3e00002 E 50 A 2,0x3e00003
```

The example specifies the following about an event with an event code of 0x3e00002:

- It is logged in the Distributed Data Manager (DDM) database by the Archive Manager for historical and reporting purposes.
Note: Events for a model that are not logged in the DDM database are displayed on the Events tab in OneClick only if they are generated while the Events tab for that model is displayed.
- It has a severity level of 50.
- It generates a major (orange) alarm whose alarm cause code is 0xe00003.

More information:

[Specify an Alarm Severity](#) (see page 45)

Generating Alarms for Events Based on the Values of Event Variables

Event discriminators are references to event variables that let you to generate alarms for events based on the values of the variables.

The following syntax is used to generate an alarm using event discriminators:

```
<eventcode> E <eventseverity> A <alarmseverity>,<alarmcause>,<eventdiscriminators>
```

where <eventdiscriminators> is a comma-separated list of one or more event variable IDs that indicate the event variables to examine to determine whether to generate an alarm. These are the event variable IDs that are mapped (in the AlertMap file) to the OIDs of the variable bindings sent with the trap.

The following event map serves as an example:

```
0x3b10011 E 70 A 1,0x3b10011,1,3
```

This example specifies that when an event with an event code of 0x3b10011 is generated, if an existing alarm with an alarm cause code of 0x3b10011 already exists on the model, another alarm for the same event is generated only if the values for *both* event variables 1 and 3 are *different* in the new event when compared to current alarms on the model generated from the same event.

The values of event variables are also stored in the alarms that are generated based on events. This means you can also use event discriminators to differentiate multiple occurrences of an alarm and clear alarms based on the values of the variables.

More information:

[Using Event Variable Discriminators to Generate Alarms](#) (see page 50)

[Clearing Alarms](#) (see page 124)

Generating Alarms Unconditionally for Each Event

By default, if an alarm exists on a model for a given event, CA Spectrum does not generate a new alarm each time the same event occurs. However, the U flag can be used to change the default behavior and generate a new alarm unconditionally each time the same event occurs. The syntax is as follows:

```
<eventcode> E <eventseverity> A <alarmseverity>,<alarmcause>,U
```

This following event map serves as an example:

```
0x3dc0000 E 50 A 1,0x3dc0000,U
```

Generating Alarms That Users Cannot Clear

By default, users can clear alarms. However, the N flag can be used to change the default behavior so that an alarm cannot be cleared by users. The syntax is as follows:

```
<eventcode> E <eventseverity> A <alarmseverity>,<alarmcause>,N
```

The following event map serves as an example:

```
0x3dc0000 E 50 A 1,0x3dc0000,N
```

Generating Alarms That Are Not Persistent

By default, alarms are persistent, that is, they are retained in memory if the SpectroSERVER is shut down and restarted. However, the T flag can be used to change the default behavior so that an alarm is not persistent. The syntax is as follows:

```
<eventcode> E <eventseverity> A <alarmseverity>,<alarmcause>,T
```

The following event map serves as an example:

```
0x3dc0000 E 50 A 1,0x3dc0000,T
```

Combining the U, N, and T Flags

Any combination of the U, N, and T flags can be used within an event map.

As an example, the following event map generates a unique alarm for each event that has an event code of 0x3dc0000. The alarm cannot be cleared by users, and it is not persistent.

```
0x3dc0000 E 50 A 1,0x3dc0000,U,N,T
```

The N flag and the T flag can be used with event discriminators as shown in the following example, which is a variation of the event map described in [Generating Alarms for Events Based on the Values of Event Variables](#) (see page 121).

```
0x3b10011 E 70 A 1,0x3b10011,1,3,N,T
```

More information:

[Using Event Variable Discriminators to Generate Alarms](#) (see page 50)

Specify an Event Frequency

The F flag can be used to detect if an event is occurring with abnormal frequency within a specified period of time. Use event rate rules to generate events for this purpose. The syntax described in this section is supported only to preserve compatibility with EventDisp files created for versions of CA Spectrum prior to version 6.5.

The syntax for using the F flag in an event map is as follows:

```
<EventCode> E <EventSeverity> F <OccurrenceLimit> <Duration> <FrequencyEventCode>
```

F

Indicates how frequently the event is evaluated.

<OccurrenceLimit>

Specifies the number of times that the event must occur.

<Duration>

Specifies the amount of time in seconds in which the number of events specified in <OccurrenceLimit> must occur to generate an event.

<FrequencyEventCode>

Specifies the event code of the event to generate if the specified number of events occurs within the specified period of time.

Specify an Event Duration

You can use the D flag to detect that a second event in what was expected to be a pair of events did not occur within a specified period of time.

Note: Use event pair rules to generate events for this purpose. The syntax described in this section is supported only to preserve compatibility with EventDisp files created for versions of CA Spectrum prior to version 6.5.

The syntax for using the D flag in an event map is as follows:

```
<FirstEventCode> E <EventSeverity> D <SecondEventCode> <Duration>  
<DurationEventCode>
```

D

Indicates that the duration of the time between events is evaluated.

<SecondEventCode>

Specifies the event code of the second event that should occur.

<Duration>

Specifies the amount of time in seconds to wait for the second event to occur.

<DurationEventCode>

Specifies the event code of the event to generate if the second event does not occur within the specified period of time.

More information:

[EventPair Rule](#) (see page 130)

Clearing Alarms

You can specify that an event should clear any of the following types of alarms:

- An alarm with a specific alarm cause code that was created without event discriminators.
- All alarms with a specific alarm cause code that were created based on event discriminators if their variable values *match* those in the alarm-clearing event.
- All alarms with a specific alarm cause code that were created based on event discriminators *regardless* of whether their variable values match those in the alarm-clearing event.

More information:

[Clear Alarms Created Without Event Discriminators](#) (see page 124)

[Clear Alarms Based on Event Discriminator Values](#) (see page 125)

[Examples of Event Maps That Clear Alarms](#) (see page 127)

Clear Alarms Created Without Event Discriminators

The following syntax is used to clear an alarm that was not created with event discriminators (which means there is only one instance of the alarm, and it does not contain copies of the event variables):

```
<eventcode> E <eventseverity> C <alarmtobecleared>
```

<eventcode>

Specifies the event code of the event.

E

Indicates the event data should be logged by the Archive Manager to the Distributed Data Manager (DDM) database.

<eventseverity>

Specifies the relative severity of the event on a scale of 0 to 100, where 0 is the least severe and 100 is the most severe.

C

Indicates that an alarm should be cleared when the specified event occurs.

<alarmtobecleared>

Specifies the alarm cause code of the alarm to clear.

Example:

```
0xfffff0000 A 1, 0xfffff0000  
0xfffff0001 C 0xfffff0000
```

The first event map generates alarm 0xffff0000 when event 0xfffff0000 occurs. The second event map clears alarm 0xffff0000 when event 0xfffff0001 occurs.

More information:

[File Syntax of Event Disposition Files](#) (see page 118)

[Examples of Event Maps That Clear Alarms](#) (see page 127)

[Clear Alarms Based on Event Discriminator Values](#) (see page 125)

[Clearing Alarms Regardless of Event Discriminator Values](#) (see page 128)

Clear Alarms Based on Event Discriminator Values

In the context of clearing alarms, event discriminators are the IDs of the event variables to examine in the alarm-clearing event and an alarm to determine whether to clear the alarm as a result of the event.

Recall that the event variable IDs are mapped (in the AlertMap file) to the OIDs of the variable bindings sent with the trap, and their values are copied from the alarm-generating event to the alarm if the alarm is generated as a result of examination of their values.

Because the values of event variables are also stored in the alarms that are generated based on events, you can use event discriminators to differentiate multiple occurrences of an alarm and clear alarms based on the values of the variables. More specifically, the alarm can be cleared if the values in the alarm-clearing event *match* the values stored in the alarm.

The following syntax is used to clear an alarm based on event discriminator values:

```
<eventcode> E <eventseverity> C <alarmtobecleared>,<eventdiscriminators>
```

<eventcode>

Specifies the event code of the event.

E

Indicates that the event data should be logged by the Archive Manager to the Distributed Data Manager (DDM) database.

<eventseverity>

Specifies the relative severity of the event on a scale of 0 to 100, where 0 is the least severe and 100 is the most severe.

C

Indicates that an alarm should be cleared when the specified event occurs.

<alarmtobecleared>

Specifies the alarm cause code of the alarm to clear.

<eventdiscriminators>

Specifies a comma-separated list of one or more event variable IDs that indicate the variables to examine to determine whether to clear the alarm.

Example:

```
0xfffff0000 A 1, 0xffff0000, 1, 2  
0xfffff0001 C 0xffff0000, 1, 2
```

Alarm 0xffff0000 is cleared only if the values of event variables 1 and 2 in the alarm-clearing event (0xffff0001) have the same values as those stored in the alarm, which are copied from the alarm-generating event (0xffff0000) when the alarm is generated. If the alarm-clearing event does *not* contain these event variables, the alarm is not cleared.

You can also use the following syntax to specify that a single event should clear multiple types of alarms:

```
<eventcode> C <alarmtobecleared>,<eventdiscriminators> C  
<alarmtobecleared>,<eventdiscriminators>,...
```

More information:

[File Syntax of Event Disposition Files](#) (see page 118)

[Examples of Event Maps That Clear Alarms](#) (see page 127)

Examples of Event Maps That Clear Alarms

In the following example, the first event map specifies that event 0x3dc0004 generates alarm 0x3dc0001. The second event map specifies that event 0x3dc0002 clears alarm 0x3dc0001.

```
0x3dc0004 E 50 A 2,0x3dc0001
0x3dc0002 C 0x3dc0001
```

The following two event maps are the same as those above except both use event discriminators. The first event map specifies that a new alarm is generated for each 0x3dc0004 event that has unique values for *both* event variables 1 and 3. The second event map clears all alarms with the alarm cause code of 0x3dc0001 if the values in the alarm for variables 1 and 3 *match* the values for the same variables in event 0x3dc0002.

```
0x3dc0004 E 50 A 1,0x3dc0001,1,3
0x3dc0002 C 0x3dc0001,1,3
```

If the event map that creates the alarm uses event discriminators, but the event map that clears the alarm does not, the event discriminators are still considered when clearing the alarm.

The following event maps use the same logic as the example above even though the event discriminators are not explicitly stated in the event map that clears the alarm. Collectively, the event maps specify that all alarms with an alarm cause code of 0x3dc0001 should be cleared if the values in the alarm for variables 1 and 3 *match* the values for the same variables in event 0x3dc0002.

```
0x3dc0004 E 50 A 1,0x3dc0001,1,3
0x3dc0002 C 0x3dc0001
```

The following event map uses the U flag to generate a unique alarm each time that event 0x3dc0004 occurs. The event that clears the alarm, event 0x3dc0002, clears all alarms that have an alarm cause code of 0x3dc0001.

```
0x3dc0004 E 50 A 1,0x3dc0001,U
0x3dc0002 C 0x3dc0001
```

In the following example, the first two event maps generate alarms. The third event map uses event 0x3dc0009 to clear alarms with different alarm cause codes. All alarms with an alarm cause code of 0x3dc00010 are cleared, and all alarms with an alarm cause code of 0x3dc00011 are cleared if their values for variables 1 and 3 match the values for the same variables in event 0x3dc0009.

```
0x3dc0006 E 50 A 2,0x3dc00010
0x3dc0007 E 50 A 1,0x3dc00011,1,3
0x3dc0009 C 0x3dc00010 C 0x3dc00011,1,3
```

Clearing Alarms Regardless of Event Discriminator Values

Use the following syntax to clear all instances of an alarm that were generated based on event discriminators regardless of whether the values in the alarm-clearing event match the values stored in the alarm instances:

```
<eventcode> E <eventseverity> C <alarmtobecleared>,A
```

<eventcode>

Specifies the event code of the event.

E

Indicates that the event data should be logged by the Archive Manager to the Distributed Data Manager (DDM) database.

<eventseverity>

Specifies the relative severity of the event on a scale of 0 to 100, where 0 is the least severe and 100 is the most severe.

C

Indicates that an alarm should be cleared when the specified event occurs.

<alarmtobecleared>

Specifies the alarm cause code of the alarm to clear.

A

Specifies that all alarm instances should be cleared regardless of the values of the variables stored in the events.

Example:

```
0xffff0000 E 50 A 1,0xffff0000,1,2,3
0xffff0002 E 50 C 0xffff0000,A
```

In this example, all instances of alarm 0xffff0000 are cleared when event 0xffff0002 occurs regardless of whether event 0xffff0002 contains event variables 1, 2, and 3, and, even if it does, regardless of whether their values match those stored in the alarms.

More information:

[File Syntax of Event Disposition Files](#) (see page 118)

About Defining Event Rules

Event rules let you to specify a complex decision making system to indicate how an event should be processed. An event rule looks for a series of events to occur on a model in a certain pattern or time frame. If the events occur as the rule specifies, another event is generated for the given model, and that new event must be defined in the EventDisp file so it can be processed appropriately by CA Spectrum.

Important! While you can create event rules manually, it is strongly recommended that you use the Event Configuration application to do so. This section is provided merely as a reference of the underlying syntax used to define rules in EventDisp files.

Event Rule Syntax

Event rules are implemented via event maps using the following syntax:

```
<EventCode> E <EventSeverity> R {<event_discriminators>} <event_rule_name>,
<event_rule_parameter1>, ...<event_rule_parameterN>
```

R

Indicates that an event rule is used.

{<event_discriminators>}

(Optional) Comma-separated list of one or more event variable IDs that indicate the variables to examine to determine whether to generate the rule output event. These are the event variable IDs that are mapped (in the AlertMap file) to the OIDs of the variable bindings sent with the trap.

Note: Event discriminators apply to each contributing event in the event rule. For examples of using event discriminators with event rules, see EventRateCounter Rule and [EventCombo Rule](#) (see page 134).

<event_rule_name>

Is expressed using the following syntax:

CA.<RuleName>

<RuleName> specifies one of the following types of rules:

- EventPair
- EventRateWindow

- EventRateCounter
- EventSequence
- EventCombo
- EventCondition
- EventCounter
- Heartbeat
- SingleEvent
- SoloEvent

<eventruleparameter>

Varies depending on which type of rule is being used.

More information:

[EventRateCounter Rule](#) (see page 132)

[EventCombo Rule](#) (see page 134)

EventPair Rule

The syntax of an event pair rule is as follows:

```
<FirstEventCode> R CA.EventPair, <SecondEventCode>, <GeneratedEventCode>, <time>
```

For example, the following event pair rule generates event 0x0001002f when event 0x0001002a occurs but is not followed by event 0x0001002b within 60 seconds:

```
0x0001002a R CA.Eventpair, 0x0001002b, 0x0001002f, 60
```

To also log event 0x0001002a and assign it a severity level of 50, the following syntax is used:

```
0x0001002a E 50 R CA.Eventpair, 0x0001002b, 0x0001002f, 60
```

More information:

[Event Pair Rules](#) (see page 62)

EventPairTimeAttr Rule

The EventPairTimeAttr Rule is similar to EventPair Rule. The difference is that it the Attribute ID is the third parameter instead of the Time window.

The syntax of an event pair rule is as follows:

```
<FirstEventCode> R CA.EventPairTimeAttr, <SecondEventCode>, <GeneratedEventCode>,
<Attribute ID>
```

Attribute ID

Specifies an attribute on the current model (the one where the event was generated one), which holds the time value for the pair rule. It can be used to set individual, model-specific time values for the rule instead of globally defined hardcoded ones.

EventRateWindow Rule

An EventRateWindow rule is an event rate rule that uses a *sliding window* (see definition on page 165) of time.

The syntax of an EventRateWindow rule is as follows:

```
<TriggerEventCode> R CA.EventRateWindow, <NumberofOccurrences>, <time>,
<GeneratedHighRateEventCode>, <GeneratedLowRateEventCode>
```

where <GeneratedHighRateEventCode> is the event to generate if the trigger event occurs at the specified frequency in the specified time frame. If you also specify an event for <GeneratedLowRateEventCode>, which is an *optional* parameter, if the frequency drops below the threshold, the rule generates the low rate event. The rule generates the high rate event again only when the frequency threshold is crossed again.

As an example, the following EventRateWindow rule generates event 0x0001002f if five events of type 0x0001002a occur within 60 seconds:

```
0x0001002a R CA.EventRateWindow, 5, 60, 0x0001002f, 0x000100030
```

Once the frequency threshold is crossed and the high rate event (0x0001002f) is generated, another rule output event is not generated until the frequency drops below 5 events within 60 seconds. At this point, the low rate event (0x000100030) is generated. The high rate event is not generated again until the frequency threshold is crossed again.

More information:

[Event Rate Rules](#) (see page 62)

EventRateWindowAttrParams Rule

An EventRateWindowAttrParams rule is an event rate rule that uses a *sliding window* (see definition on page 165) of time.

The syntax of an EventRateWindowAttrParams rule is as follows:

```
<TriggerEventCode> R CA.EventRateWindow, <Attribute containing  
NumberOfOccurrences>, <Attribute containing time window>,  
<GeneratedHighRateEventCode>, <GeneratedLowRateEventCode>,  
<GeneratedStopEventCode>
```

where <GeneratedStopEventCode> is the event you generate to stop the rule. Update the attributes for the model and then generate this stop event to stop the rule instance on that model. The next rate window event starts a new rule instance, which reads and uses the new attribute values.

The EventRateWindowAttrParams rule is similar to the EventRateWindow rule. However, the EventRateWindowAttrParams rule accepts attribute ids instead of direct values for the event occurrence parameter and the time window parameter. These attributes must be present on the model where the event is generated and should contain integer values.

Note: See EventRateWindow Rule (see page 131) for <GeneratedHighRateEventCode> and <GeneratedLowRateEventCode> event descriptions.

More information:

[Event Rate Rules](#) (see page 62)

EventRateCounter Rule

An EventRateCounter rule is an event rate rule that uses a *sequential window* (see definition on page 165) of time.

The syntax of an EventRateCounter rule is as follows:

```
<TriggerEventCode> R CA.EventRateCounter, <NumberOfOccurrences>, <time>,  
<GeneratedEventCode>
```

As an example, the following EventRateCounter rule generates event 0x0001002f if 5 events of type 0x0001002a occur within 60 seconds:

```
0x0001002a R CA.EventRateCounter, 5, 60, 0x0001002f
```

To also log event 0x0001002a and assign it a severity level of 50, the following syntax is used:

```
0x0001002a E 50 R CA.EventRateCounter, 5, 60, 0x0001002f
```

Example: Using Event Discriminators with an EventRateCounter Rule

The following example shows two event discriminators used in an EventRateCounter rule:

```
0x10001 E 50 R {1,2} CA.EventRateCounter, 3, 60, 0xffff0000
```

The event discriminator list, {1,2}, contains variable IDs 1 and 2. Therefore, in order for event 0xffff0000 to be generated, event 0x10001 must occur 3 times within 60 seconds, and all 3 instances must contain the same values for variable IDs 1 and 2.

For example, if event 0x10001 occurred 3 times in 60 seconds, and each time variable ID 1 had a value of 10.253.40.57 and variable ID 2 had a value of 65, then event 0xffff0000 would be generated. However, if event 0x10001 occurred 3 times in 60 seconds but the first 2 times variable ID 1 had a value of 10.253.30.57 and the third time it had a value of 10.253.89.60, then 0xffff0000 would not be generated.

More information:

[Using Event Variable Discriminators to Generate Alarms](#) (see page 50)

[Event Rate Rules](#) (see page 62)

EventSequence Rule

An EventSequence rule is an event series rule that requires the series of events to occur in a specific sequence.

The syntax for the EventSequence rule is as follows:

```
<FirstEventCode> R CA.EventSequence, <GeneratedEventCode>, <time>,
<SecondEventCode>, <ThirdEventCode>,...<NthEventCode>
```

As an example, the following EventSequence rule generates event 0x0001002f when events 0x0001002a, 0x0001002b and 0x0001002c occur, in that order, within 60 seconds.

```
0x0001002a R CA.EventSequence, 0x0001002f, 60, 0x0001002b, 0x0001002c
```

To also log event 0x0001002a and assign it a severity level of 50, the following syntax is used:

```
0x0001002a E 50 R CA.EventSequence, 0x0001002f, 60, 0x0001002b, 0x0001002c
```

More information:

[Event Series Rules](#) (see page 64)

EventCombo Rule

An EventCombo rule is an event series rule that requires the series of events to occur but the order of occurrence does not matter.

The syntax of an EventCombo rule is as follows:

```
<FirstEventCode> R CA.EventCombo, <GeneratedEventCode>, <time>, <EventCodeA>, <EventCodeB>, ...<EventCodeN>
```

As an example, the following EventCombo rule generates event 0x0001002f if event 0x0001002a occurs, and it is followed by at least one instance each of event 0x0001002b and event 0x0001002c within 60 seconds and in any order.

```
0x0001002a R CA.EventCombo, 0x0001002f, 60, 0x0001002b, 0x0001002c
```

To specify that the new event should be generated but one of several events can first trigger the rule, create a series of n rules where n is the number of events in the combination, and where each rule uses a different event for the trigger. For example, consider the following three rules:

```
0x0001002a R CA.EventCombo, 0x0001002f, 60, 0x0001002b, 0x0001002c
0x0001002b R CA.EventCombo, 0x0001002f, 60, 0x0001002a, 0x0001002c
0x0001002c R CA.EventCombo, 0x0001002f, 60, 0x0001002a, 0x0001002b
```

Using this set of rules, any combination of events 0x0001002a, 0x0001002b, and 0x0001002c occurring at least once within 60 seconds and in any order would generate event 0x0001002f.

To also log event 0x0001002a and assign it a severity level of 50, the following syntax can be used:

```
0x0001002a E 50 R CA.EventCombo, 0x0001002f, 60, 0x0001002b, 0x0001002c
```

Example: Using Event Discriminators with the EventCombo Rule

The following example shows two event discriminators used in an EventCombo rule:

```
0x10001 E 50 R {1} CA.EventCombo, 0xffff0000, 60, 0x10002
```

The event discriminator list, {1}, contains variable ID 1. Therefore, event 0x10001 must occur, and then event 0x10002 must occur within 60 seconds, and each must contain the same values for variable ID 1 in order for event 0xffff0000 to be generated.

For example, if event 0x10001 occurred and had 10.253.40.57 as a value for variable ID 1, and event 0x10002 occurred 45 seconds later and had a value of 10.253.40.57 for variable ID 1, then event 0xffff0000 would be generated. However, if event 0x10001 occurred and had 10.253.40.50 as a value for variable ID 1, and event 0x10002 occurred 45 seconds later and had a value of 10.253.40.57 for variable ID 1, then event 0xffff0000 would not be generated.

More information:

[Event Series Rules](#) (see page 64)

EventComboInclusive Rule

The EventComboInclusive rule is similar to the EventCombo rule. The difference is that it registers all the combo events, and not just the one which the rule is defined for.

The syntax of an EventComboInclusive rule is as follows:

```
<FirstEventCode> R CA.EventComboInclusive, <GeneratedEventCode>, <time>,
<EventCodeA>, <EventCodeB>, ...<EventCodeN>
```

As an example, the following EventComboInclusive rule generates event 0x0001002f if either of event 0x0001002a, 0x0001002b or event 0x0001002c occurs:

```
0x0001002a R CA.EventComboInclusive, 0x0001002f, 60, 0x0001002b, 0x0001002c
```

More information:

[EventCombo Rule](#) (see page 134)

EventCondition Rule

The conditional expressions in an EventCondition rule can compare a variable binding value or a CA Spectrum attribute value to a user-specified value using standard comparison operators as follows:

```
<FirstEventCode> R CA.EventCondition, "<conditional expression 1>", <event to
generate when 1 is TRUE>,"<conditional expression n>", <event to generate when n is
TRUE>, "default", <default event>
```

"<conditional expression x>"

Consists of one or more expressions comparing a variable binding value or a CA Spectrum attribute value to a user-defined value (x represents any value from 1 to n).

<event to generate when x is TRUE>

Specifies the event that is generated if the conditional expression evaluates to TRUE.

"default", <default event>

(Optional) Specifies a default event that is generated if none of the conditions are met. For example, if the following syntax was included at the end of the rule, event 0xffff1234 would be generated if none of the other conditions expressed in the rule were met:

"default", 0xffff1234

Note: Default can be expressed as "DEFAULT", "default", or "Default".

Conditional expressions are evaluated from left to right and, with some simplification, follow C programming-style evaluations. If the whole condition evaluates to TRUE, then the event is generated.

Condition Syntax

The conditional text is always enclosed in quotation marks as follows:

"condition"

A simple condition is made up of data or objects and comparison operators or methods.

Data or Objects

Each data element or object to be compared in the condition must be contained within curly brackets and must have both a type and a value:

```
{ TYPE VALUE }
```

For example, a comparison that involves an integer value of 2 is expressed as { I 2 }.

The following lists the supported types and their meanings:

Short Symbol	Alternate Names	Meaning	Type Values	Examples
A	Addr ADDR address IP Addr IP Address IP_ADDRESS	Contains an IP address	XXX.XXX.XXX.XXX where each XXX subterm is a number from 0 to 255, and the whole term forms a valid IP address	{ Addr 192.168.1.1 }
a	attr ATTR attribute ATTRIBUTE	References a model attribute of the model for which the event rule is being processed. When evaluated, the attribute's current value is read. The attribute's type is used to determine if comparison is valid.	An attribute ID, specified as a hex number. The leading 0x is optional. The letters a-f may be lower- or uppercase. The reading of table attributes using an object ID or variable data as an index is also supported (see examples).	{ attr 0x11564 } { Attribute A00044 } { attr 0xffff0001 obj 1.1.6.8.0.1 } would read table attribute 0xffff0001, with "1.1.6.8.0.1" as the OID suffix (index). { attr 0xffff0001 VARDATA 2 } would read table attribute 0xffff0001 using the object id contained in the second variable binding as the OID suffix (index).
B	BOOL Bool Boolean boolean	A boolean value	False, true This value is not case-sensitive	{ B True } { boolean false }

Short Symbol	Alternate Names	Meaning	Type Values	Examples
H	HEX Hex Hex_ID HEX Id	A hex attribute ID value (this is just a value, not an attribute reference, use 'a' for that purpose)	An attribute ID, specified as hex number. The leading 0x is optional. The letters a-f may be lowercase or uppercase.	{ Hexid 0xffff0123 } { H ABCD } { HEX 0X91 }
I	Int integer INTEGER INT	An integer value	Any number within the range of -214783648 to 214783647	{ Int 10 } { I 98765 } { integer -300 }
L	Unsigned long int LONG long UNSIGNED_LONG_INTEGER LongInt	An unsigned long 64 bit integer	Any number within the range of 0 to 18446744073709551615	{ L 0 } { LongInt 123456789098 }
o	Object ID obj obj_id OBJECT	An object ID	X X.X X.X.X and so on where X is an unsigned integer (>= 0)	{ o 1.2.3.4.5.6 } { object_id 1.3.6.1.2.1 } { OBJ 100000.4.5 }
O	Octet Octet String Decimal Octet String DEC Oct_Str Dec_OCTET_str	A tagged octet string comprised of decimal values	#. #. #. #... where # is any number between 0 and 255	{ Oct Str 1.2.3.4 } { OCTET_STR 255.0.1.20 } { OCTSTR 10.20.30.40.50 }
R	REAL real Real	A real number (double)	Any number containing a '.', and possibly followed by an exponent: E e + - EXP EXP any number	{ R 1. } { Real 3.1415 } { REAL -2.843E-17 } { R .00001e+20 }
S	String str Str STRING	A string	Any characters enclosed in double quotes	{ S \"a string\" } { String \"another string\" } { str \"12345a@b?c*\" }

Short Symbol	Alternate Names	Meaning	Type Values	Examples
				{ S \" } }
U	unsigned unsigned long unsigned long int ULONG U INT U_long_integer	An unsigned long integer	Any number in the range from 0 to 4294967295	{ uint 1234567890 } { UNSIGNED INT 0 }
v	variable data VARDATA Var_DATA EventAttr event_attr	References an event attribute (variable data) from the current event. Like a model attribute reference, this is evaluated when needed, and the event attribute type is used to check if the comparison is valid	A unsigned integer (> 0)	{ v 1 } { VARDATA 2 } { event attr 3 }
X	Hex octet string HEXOCT_STR Hex_Octet HEXOctet	A tagged hexadecimal octet string	XX.XX.XX..... Where XX is a hex number from 0 to FF	{ X 12.01.AB.EF } { HEXOCT AB.CD.EF.01 } { Hex octet string 2.3 } { X a.b.c }

Using the Escape Character

In addition to enclosing the entire condition within quotation marks, you must also enclose a string in quotations marks. Also, to help ensure that the entire condition is parsed properly, the opening and closing quotation marks enclosing the string both must be preceded by a backslash. The following condition, which includes string xyz, serves as an example:

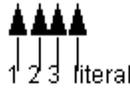
```
"{ S \"xyz\" } == { v 1 }"
```

If you want to include a backslash '\ ' or a double quote in the string itself, you need 3 backslashes before each of these characters, as shown in the following example where:

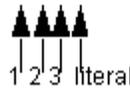
- Backslash 3 represents the escape for the literal backslash or quote
- Backslash 2 represents the escape character needed to escape backslash 3 within the string
- Backslash 1 represents the escape character needed to escape backslash 2

This same logic holds true for a double quote example.

```
"{ S \"backslash character: \\\" example \" } == { attr 1 }"
```



```
"{ S \"quote character: \"\" example \" } == { attr 1 }"
```



Regular expressions also use the backslash '\ ' as an escape character. Because the regular expression can be used in a condition, the backslash used within the regular expression as the escape character must be preceded by several backslashes.

Comparing Strings

The strcmp method is used to compare characters in a string (for example, in strings, octet strings, IP Addresses, and so on). Note that the implementation of this method differs slightly from the standard C implementation of strcmp. This method returns TRUE if the two strings are equal, and it returns FALSE if the strings are not equal.

To make use of this method, use the following format:

```
"strcmp({ TYPE <string> }, { TYPE <string> })"
```

For example, the following condition compares an IP Address with a CA Spectrum attribute whose value is an IP Address. If the two strings are the same, the condition returns TRUE.

```
"strcmp({ A 179.82.253.01 }, { attr 0x00011aec })"
```

Note the following in the example:

- Each type/string pair to be compared is enclosed in curly brackets
- The two type/string pairs are separated by a comma and enclosed in parenthesis
- The name of the method, `strcmp`, is placed on the left hand side of the parenthesis
- The entire condition is enclosed in quotes

Regular Expressions

The `regexp` or `REGEXP` method is used to compare a string to an input pattern. To do this, regular expressions use a series of meta-characters that let you express the pattern of characters that you are looking for. The method returns `TRUE` if the regular expression input pattern matches the input string; otherwise, it returns `FALSE`.

To make use of this method, use the following format:

```
"regexp({ TYPE <string> }, { TYPE <input pattern> })"
```

The `EventCondition` rule supports the syntax of the Perl Compatible Regular Expression (PCRE) package. The following are some of the basic meta-characters supported by the PCRE package and examples of their usage:

Meta-character	Meaning	Example
<code>^</code>	Indicates the beginning of a line.	The following example searches for the string "CA" occurring at the beginning of a line in event variable 1: "regexp({ VARDATA 1 }, { S \"^CA\" })"
<code>\$</code>	Indicates the end of a line.	The following example searches for lines ending with the string "CA" in event variable 1: "regexp({ VARDATA 1 }, { S \"CA\$\" })"
<code>[]</code>	Encloses a character class. A character class shows some literal text that you would like to let at a certain point within the string.	This example searches for the string "port" followed by a value of 1, 2 or 3 in event variable 1: "regexp({ VARDATA 1 }, { S \"port [1-3] \" })"
<code>*</code>	Indicates zero or more of the specified preceding characters.	This example searches for zero or more occurrences of 172 in the value of the attribute 0x00011aec: "regexp({ attr 0x00011aec }, { S \"(172)*\" })"

+	Indicates one or more of the specified preceding characters.	This example searches for one or more occurrences of 172 in the value of the attribute 0x00011aec: "regexp({ attr 0x00011aec }, { S \"(172)+\" })"
.	Represents any single character except for a new line character.	This example searches for an occurrence of the word "port" followed by a space and then any single character within event variable 1: "regexp({ VARDATA 1 }, { S \"port . \" })"
	Separates alternative patterns.	"regexp({ VARDATA 1 }, { S \"interface port\" })" This example searches for either the word interface or the word port within the event variable 1.
\	Used as a general escape character letting you to use the literal meaning of a meta-character. When you use the escape character within the regular expression, you must be sure to also consider the necessary escape characters to be used within the context of the string and the condition..	This example searches for the string "172.55" within event variable 1: "regexp({ VARDATA 1 }, { S \"172\\\\.55\" })" Because the . character is usually treated as a meta-character within a regular expression, it is necessary to use the escape character (a backslash) to indicate that you would like the . to be treated literally. At the regular expression level, this yields the following syntax: 172\\.55. However, since you are using this regular expression within a string, you must precede the backslash with an additional backslash. Reading the expression from left to right, the first backslash represents the escape character needed to escape the second backslash within the string. This yields the following syntax: 172\\\\.55. Additionally, you are using the string within the context of the condition, therefore each existing \ must have a corresponding backslash to be used as an escape. This yields the following syntax: 172\\\\\\\\.55.

Comparison Operators

The following comparison operators are supported for numeric or boolean values:

==

equal to

!=

not equal to

>

greater than

<

less than

<=

less than or equal to

>=

greater than or equal to

Most numeric values can be compared with each other even if they are not of the same type. For example, the following condition compares the integer 2 to the value of a CA Spectrum attribute:

```
"{ I 2 } == { attr 0x000117dc }"
```

Exists Operator

You can use the exists operator to check for the existence of a variable binding value or other value:

```
exists( <expression> )
```

This can be useful, for example, when you want to evaluate a value if it exists, and exit the event condition rule if it does not. The following spelling variants are supported:

exists

Exists

EXISTS

An exists conditional expression returns TRUE when the expression is valid and contains a value. For example, "exists({ v 1 })" returns TRUE when event variable 1 exists and contains any value.

To terminate an event condition rule without action when an exists conditional expression returns FALSE, you can use the “no action” action (which sends the 0x00010000 null event). The following spelling variants are supported in either all lower case, all upper case, or initial capital letters:

no action

no-action

no_action

As an example, the following event condition rule checks whether event variable 1 exists in event 0xffff0000. If it does not exist, no action is taken. If it does exist, event 0xffff0001 is generated.

```
0xffff0000 E 50 R CA.EventCondition,      \  
    " ! Exists( { v 1 } )", "No-Action",    \  
    " { v 1 } == { I 1 } ", 0xffff0001
```

The Logical NOT (!)

You can use the logical NOT (!) operator to reverse the logical value of an expression. Apply the logical NOT (!) in the same way as you would when writing C++ code. For example, the following condition compares variable binding 1 and variable binding 2. The logical NOT (!) is applied to the outcome of the comparison. Thus, if variable binding 1 is equal to variable binding 2, the entire expression evaluates to false.

```
“! ( { v 1 } = = { v 2 } )”
```

Complex Conditions

More complex conditions can be created which use logical operators and parenthesis to combine simple conditions. Valid logical operators are the following:

&&, which represents AND

||, which represents OR

The following condition includes several subconditions enclosed in braces and linked together using logical operators.

```
“({ I 2 } = = { I 2 } ) && ( { I 2 } != { I 3 } )”
```

For this condition to evaluate to TRUE, both of the subconditions on either side of the && must evaluate to TRUE. Since an integer value of 2 is equal to an integer value of 2, the left side of the condition is TRUE. Since an integer value of 2 is not equal to an integer value of 3, the right side of the condition also evaluates to TRUE. This means that the whole condition evaluates to TRUE.

```
"({ I 3 } = { I 4 } ) || ( { I 4 } > { I 2 } )"
```

For this condition to evaluate to TRUE, the subcondition on the left hand side of the || or the subcondition on the right hand side of the || must evaluate to TRUE. Since 3 is not equal to 4, but 4 is greater than 2, the entire expression evaluates to TRUE.

Multiple subconditions can be used to create the necessary expression. For example, the following condition evaluates to TRUE since 4 is greater than 2, and 3 is less than 8.

```
"({ I 3 } = { I 2 } || { I 4 } > { I 2 } ) && ( { I 3 } < { I 8 } || { I 2 } = { I 4 } )"
```

Nested Conditions

Although simple conditions can be combined together using logical operators to create complex conditions, you cannot use simple conditions as a part of other expressions. A simple condition can have only one comparison operator in it.

For example, the following syntax is not supported. The result of a strcmp() cannot be used as the argument for an equals (==) operator:

```
"strcmp ( { S \"a\" }, { S \"a\" } ) = { B TRUE }"
```

Example: Basic EventCondition Rule

The following example shows a basic EventCondition rule that uses some of the condition syntax described in the previous sections. The first condition that evaluates to TRUE is used, and the event code immediately following that condition is generated.

```
0x00045678 R CA.EventCondition,
    \
    "regex({ VARDATA 1 }, { S \"port [1-3] \"})",
    \
    0x00012345, "{ VARDATA 2 } = { attr0x000117dc }",
    \
    0x00012344, "strcmp({ A 179.82.253.01 }, \{ attr 0x00011aec } \
    \
    )",
    \
0x000122334
```

Example: (Complex EventCondition Rule) Generating an Event Based on a Variable Binding Value

The following example shows an EventCondition rule that generates an alarm based on the value of one of the variable bindings sent with the trap. The event condition rule generates a second event, 0xffff0000, if the value of variable binding 1 is equal to any of the following values: DAT0005, DAT0006, DAT0007, DAT0008, DAT0011, DAT0012, DAT0013, DAT0014, DAT0021, DAT0022, or DAT0023.

```
0x1030f E 50 R CA.EventCondition,          \
  "regex( { v 1 },                          \
    { S \"DAT00(05|06|07|08|11|12|13|14|21|22|23) \" } \
  )",                                         \
  "0xffff0000 -:-"
```

There is a space at the end of the regular expression, which indicates that each pattern must end with a space.

The regular expression first looks for a match between variable binding 1 and DAT00 combined with any of the choices in the brackets (05 , 06 , 07 , etc.).

There is a -:- at the end of the event condition. This symbol specifies that all of the variable binding values from the originating event (0x1030f) should be copied to the new event (0xffff0000).

To generate an alarm when the event condition evaluates to true, an event map for the new event (0xffff0000) that specifies to generate an alarm is needed in the EventDisp file.

More information:

[Event Condition Rules](#) (see page 61)

[Copy Event Variables from One Event to Another](#) (see page 151)

EventCounter Rule

The EventCounter rule implements a counter, counting up for 'up' and down for 'down' events. The rule creates an event if a certain threshold is reached.

The syntax of EventCounter rule is as follows:

```
<CountUpEventCode> R CA.EventCounter, <CountDownEventCode>, <threshold>,
<ThresholdBreachEventCode>, <ThresholdResetEventCode>
```

CountUpEventCode

Counts up by one. The first one also initiates the counter rule.

CountDownEventCode

Counts down by one.

threshold

The counter threshold (integer number). Once the count reaches (equals) that threshold, the target event is generated

ThresholdBreachEventCode

Generates when the count threshold is reached

ThresholdResetEventCode

Generates when the count is below the threshold again

Example: EventCounter rule

```
0x0f420001 E 50 R CA.EventCounter, 0x0f420002, 3, 0x0f420003, 0x0f420004
```

Use events 0x0f420001 to count up, and events 0x0f420002 to count down. When the counter is greater or equal to 3, event 0x0f420003 will be generated. Once the count falls below 3 again, event 0x0f420004 will be generated.

Heartbeat Rule

The Heartbeat rule is set to watch a 'heartbeat' event. The event is seen at a regular interval. In case any instance of the heartbeat is found missing, the rule creates an event.

```
<TriggerEventCode> R CA.Heartbeat, <HeartbeatEventCode>,
<MissingHeartbeatEventCode>, <timeout>, <StopEventCode>
```

TriggerEventCode

Instantiates the heartbeat event rule

HeartbeatEventCode

The heartbeat event code

MissingHeartbeatEventCode

The event generated in case a heartbeat is missing

timeout

The time gap between individual heartbeats (in seconds)

StopEventCode

(Optional) The event code that stops the rule. The rule instance is running forever, looking for a heartbeat until stopped

Example: Heartbeat Rule

```
0x0f440001 E 50 R CA.Heartbeat, 0x0f440002, 0x0f440003, 10, 0x0f440004
```

Event 0x0f440001 instantiates the heartbeat rule. Next, it will look for event 0x0f440002 to occur at least once every 10 seconds. If a heartbeat event is found missing, the rule creates event 0x0f440003. Use event 0x0f440004 to stop the heartbeat rule instance.

SoloEvent Rule

The SoloEvent rule finds an instance of the target event that is not followed by or preceded by any other event in a defined time window. The events that are in the prevent list may not occur. Other events will not affect the Solo event.

As an example, you may want to have a rule triggered when event A occurred, but only none of events B, C or D (the 'prevent' events) occurred within five minutes before or 10 minutes after it.

The syntax of the SoloEvent Rule is as follows:

```
<TriggerEventCode> R CA.SoloEvent, <StopEventCode>, <SoloEventCode>,
<prePreventPeriod>, <postPreventPeriod>, <TargetEventCode>, <PreventCode 1>,
<PreventCode 2 (optional)>, ... , <PreventCode N (optional)>
```

TriggerEventCode

Initiates the 'solo' event rule.

StopEventCode

Stops the rule. The rule will run endlessly, unless you set the stop event.

SoloEventCode

Sets the solo event.

prePreventPeriod

Sets the time period before the solo event where none of the 'prevent' events may occur (in seconds).

postPreventPeriod

Sets the time period after the solo event where none of the 'prevent' events may occur.

TargetEventCode

Defines the event that will generate when the rule triggers (when just the 'solo' event was seen).

PreventCode 1 to PreventCode N

Defines a list of 'prevent' events.

Example: SoloEvent Rule

The following example will create event code 0x0f400005 if the solo event 0x0f400003 was seen, and none of the events 0x0f400006, 0x0f400007, 0x0f400008 or 0x0f400009 were seen within 20 seconds before or after it. The first event 0x0f400001 will have to be generated to instantiate the rule. Event 0x0f400002 is available to stop the rule.

```
0x0f400001 E 40 R CA.SoloEvent, 0x0f400002, 0x0f400003, 20, 20, 0x0f400005,
0x0f400006, 0x0f400007, 0x0f400008, 0x0f400009
```

SingleEvent Rule

The SingleEvent rule reduces an event stream where one event ('up' event) may occur multiple times, before a reset ('down') event is seen. Instead of the multiple 'up' events, a single event is set that can be reused in other rules, denoting the condition ('up' or 'down').

The syntax of SingleEvent rule is as follows:

```
<TriggerEventCode> R CA.SingleEvent, <ResetEventCode>, <SingleTargetEventCode>, <SingleResetTargetEventCode>
```

TriggerEventCode

Occurs multiple times and should be converted into a 'single' occurrence. This will also trigger the rule to be instantiated when it occurs the first time.

ResetEventCode

Sets the reset event. When this event is seen, the rule is ready to create another single event again.

SingleTargetEvent

Generates the first time the trigger event is seen either when the rule is instantiated, or the first time the trigger event occurs after reset event is seen.

SingleResetTargetEvent

(Optional) Generates when the reset event is seen.

Example: SingleEvent Rule

```
0x0f430001 E 50 R CA.SingleEvent, 0x0f430002, 0x0f430003, 0x0f430004
```

Generates event 0x0f40003 every time event 0x0f430001 is seen for the first time. The SingleEvent is instantiated either at initial rule creation, each time after the trigger is seen the first time or after the reset event 0x0f430002 was seen. Event 0x0f430004 will also be generated when the reset event has been seen.

Using Multiple Event Rules in a Single EventDisp Entry

You can specify that a single event be processed using multiple event rules. For example, the following event disposition entry specifies that event 0x0001002a be processed using both the EventSequence rule and the EventPair rule:

```
0x0001002a R CA.EventSequence, 0x0001002c, 60, 0x0001002d, 0x0001002e \  
R CA.EventPair, 0x0001002b, 0x0001002f, 60
```

This entry specifies that event 0x0001002c will be generated when events 0x0001002a, 0x0001002d, and 0x0001002e occur in that order within 60 seconds, and event 0x0001002f will be generated if event 0x0001002a occurs, but is not followed by event 0x0001002b within 60 seconds.

The backslash character is used at the end of the line to show that the event disposition entry continues onto the next line.

Note: Multiple rules must be specified within a single event disposition entry. If you were to create two separate event disposition entries for an event, only the first event disposition entry would be processed.

More information:

[Syntax Errors in EventDisp Files](#) (see page 157)

Copy Event Variables from One Event to Another

When you use event rules to trigger events, depending on the rule, the event that is generated as the output of the rule might be generated only after multiple, contributing events occur or certain complex conditions are met.

By default, a rule output event does not have any event variables. However, you can specify that the values of the event variables in the events that contribute to the rule's processing should be copied to the rule output event. This lets you to specify event processing behaviors for the rule output event based those values, which you can do using event variable discriminators. Moreover, because the values of event variables in events that generate alarms are also stored in those alarms, you can also use event discriminators to specify alarm processing (generation or clearing of alarms) based on the values of the copied variables.

This section provides reference information on event variable copying syntax.

More information:

[Generating Alarms for Events Based on the Values of Event Variables](#) (see page 121)

[Clearing Alarms](#) (see page 124)

[Event Variable Copying and Event Discriminators](#) (see page 157)

Event Variable Copying Syntax

Consider the following EventCombo rule that generates event 0xa000f when events 0x10002, 0x10003, and 0x10004 are all received within 10 seconds of event 0x10001:

```
0x10001 R CA.EventCombo, 0xa000f, 10, 0x10002, 0x10003, 0x10004
```

Now assume that the contributing events have variable bindings that have values, some of which you want to copy to event 0xa000f as follows:

Contributing Event	Event Variables to Copy	Event Variables in 0xa000f to Receive the Copies
0x10001	2, 3	1, 2
0x10002	1, 5	3, 4
0x10003	none	none
0x10004	2, 3, 4, and 5	5, 6, 7, and 8

Note: An event generated by an event rule does not have any event variables unless they are copied from contributing events.

To copy the event variables as specified in the preceding table, the following event variable copying syntax is used:

```
0x10001 R CA.EventCombo, "0xa000f 2-3:1-2", \
    10, \
    "0x10002 1:3, 5:4", \
    0x10003, \
    "0x10004 2-5:5-8"
```

This syntax copies event variables 2 and 3 in 0x10001 to event variables in 0xa000f, copies event variables 1 and 5 in 0x10002 to event variables 3 and 4 in 0xa000f, and copies event variables 2, 3, 4, and 5 in 0x10004, respectively, to event variables 5, 6, 7, and 8 in 0xa000f.

Event Parameters and Variable IDs

Event parameters specify the event variables to copy from a contributing event to the event generated by the rule. This information is associated with the contributing event, and it is entered after the event ID. To specify that the event copy information is part of the event parameter and not the next rule parameter, the event ID and the copy information need to be enclosed in double quotes, as shown:

```
"0x10002 1:3"
```

Event Variable Copy Information

The event variable copy information section of an event parameter can consist of several parts, each separated by a comma. Each part specifies a source variable ID or range of IDs, followed by a target variable ID or range of IDs. A colon separates the source IDs from the target IDs, as shown:

```
"0x10001 1:1, 2:3"
```

If the source IDs and the target IDs are the same, either can be left empty.

Variable IDs

A single event variable is identified by its ID, which typically is a small number.

To copy a specific source event variable, enter its ID in the source position (left of the colon) in the copy information section. Similarly, to copy to a specific event variable, enter its ID in the target position (right of the colon).

For example, the following syntax copies variable 1 in event 0x10001 to variable 1 in the rule-generated event:

```
"0x10001 1:1"
```

Similarly, the following syntax copies variables 1 through 5 in event 0x10003 to variables with the same IDs in the rule-generated event:

```
"0x10003 1:1, 2:2, 3:3, 4:4, 5:5"
```

Variable IDs Using Ranges

Ranges of source and target variable IDs can be specified using a start ID followed by the dash ('-') character and then a stop ID. The start and stop IDs are included in the range.

The start ID, the stop ID, or both IDs can be left out. An entry with no start ID copies all of the IDs in the range from 1 to the stop ID, inclusive. An entry with no stop ID copies all of the variables with IDs equal to or greater than the start ID. An entry without either the start ID or the stop ID copies all of the variables (the dash can be left out too).

The number of variables in the source and target ranges need to be the same. If the numbers do not match, the number of IDs in the smaller range is used to copy variables, and a warning is generated.

As examples, the following 3 rule fragments are all equivalent. Each copies variables 1 through 73 in event 0x10004 to variables with the same IDs in the rule-generated event:

```
"0x10004 1 - 73 : 1 - 73"  
"0x10004 1-73 :"  
"0x10004 :1-73"
```

To copy variables 1 through 3 in event 0x10005 to the rule-generated event using new IDs 6 through 8:

```
"0x10005 1-3:6-8"
```

To copy all variables up to and including variable 5 to the rule-generated event using new IDs beginning with 1:

```
"0x10006 -5:1-"
```

To copy variables 2 through 5 to new IDs 1 through 4, and to also copy all remaining variables beginning with ID 7 using the same target IDs in the rule-generated event:

```
"0x10007 2-5:1-4, 7-:7-"
```

To copy all of the variables in event 0x10008 to the rule-generated event, use any of the following:

```
"0x10008 -:-"  
"0x10008 -:"  
"0x10008 :-"  
"0x10008 :"  
"0x10008 1-:1-"
```

The following example copies variables 3 to 5 from event 0x10009 to the rule-generated event where the source event has fewer variable entries than the target event. This example will succeed, but a warning will be generated:

```
"0x10009 3-5:3-"
```

Copying Variables from the Initial Event

Copying variables from the initial event (the event that the event disposition entry is for) is a special case. In this case, the copy information for the initial event is attached to the event to be generated rather than the source event.

As an example, consider the following EventCombo rule and EventRate rule, both of which watch for event 0x10001. If event 0x10001 occurs in combination with event 0x10002, its event variable 1 is important; if it occurs frequently, its event variable 3 is of interest:

```
0x10001      R CA.EventCombo, "0xa000f 1:1", 10, 0x10002 \
              R CA.EventRateCounter, 10, 3600, "0xa000e 3:1"
```

Different types of rules specify the event to generate in different parameter positions. In the preceding example, the EventCombo rule uses the first variable position, and the EventRateCounter rule uses the third variable position.

Multiple Event Occurrence

When two event rules watch for multiple occurrences of an event (for example, an EventRateWindow rule and an EventRateCounter rule), the final occurrence of the event is used to copy the event variables, since they are the most current.

As an example, assume that event 0x10001 is a security alert from an intrusion detection system that holds information about the severity of a potential intrusion. Also assume that the following event rate rule exists, which generates event 0xa000f when event 0x10001 is received at least 5 times during a 100 second interval:

```
0x10001 R CA.EventRateCounter, 5, 100, "0xa000f 1:1"
```

A possible sequence of contributing events might be as follows:

Note: For readability, the severity event variable is shown as a text string.

```
0x10001  Warning
```

```
0x10001  High
```

```
0x10001  Warning
```

```
0x10001  Critical
```

```
0x10001  Critical
```

where the fifth 0x10001 event received triggers the generation of the rule output event, creating event 0xa000f with its variable 1 set to "Critical."

As an alternative and more specific approach, the following EventCondition rule could be used so that the rule output event is only generated in response to five critical 0x10001 events (instead of in response to five 0x10001 events of any severity):

```
0x10001 R CA.EventCondition, \
        "regexp ( { variable 1 } , { S \"Critical\" } )", "0xa0001 1:1"
0xa0001 R CA.EventRateCounter, 5, 100, "0xa000f"
```

More information:

[Variable Descriptions and Syntax](#) (see page 34)

[Event Variable Copying and Event Discriminators](#) (see page 157)

Event Variable Copy Example

Having reviewed [Event Variable Copying Syntax](#) (see page 157) for information about how to copy event variables from contributing events to rule-generated events, examine again the following event variable copy example:

```
0x10001 R CA.EventCombo, "0xa000f 2-3:1-2", \
    10, \
    "0x10002 1:3, 5:4", \
    0x10003, \
    "0x10004 2-5:5-8"
```

When this EventCombo rule generates event 0xa000f, the event variables in the contributing events are copied as shown in the following table:

Source Event	Source Variable ID	0xa000f (Target) Variable ID
0x10001	2	1
0x10001	3	2
0x10002	1	3
0x10002	5	4
0x10004	2	5
0x10004	3	6
0x10004	4	7
0x10004	5	8

Event Variable Copying and Event Discriminators

The event copying syntax can be used in conjunction with event discriminators to differentiate events that generate alarms. For example, assume alarm 0xffff0000 should be generated when event 0x10001 is received, and event variable 1 contains the index of the board that failed. Event variable 1 can be used as the alarm discriminator. Also assume that the device sometimes falsely reports errors, and the alarm should be cleared if supporting event 0x10002 is not received within 10 seconds. The following event maps satisfy these requirements:

```
0x10001 E 50 \
    A 2, 0xffff0000, 1 \
    R CA.EventPair, 0x10002, "0xa000f 1:1", 10
0xa000f C 0xffff0000, 1
```

The EventPair rule is used to watch for the second event and issue the clearing event containing the correct variable needed to clear the alarm. Note the use of the event discriminator to determine whether to generate and to clear the alarm, as well as the copy information contained in the rule.

Syntax Errors in EventDisp Files

All of an event's processing syntax must exist as a single event map. An EventDisp file that lists the same event code twice as the first entry on a line is considered to be improperly formatted. In this situation, the first line that applies to a particular event code is processed, and any additional entries are discarded.

The following example illustrates *incorrect* syntax:

```
0x3e00002 E 50
0x3e00002 A 2,0x3e00002
```

The following example illustrates *correct* syntax:

```
0x3e00003 E 50 A 2,0x3e00003
```

In the incorrect example, only the first event map is processed. Thus, when event 0x3e00002 is received, it is logged with an event severity of 50. However, no alarm is generated.

The correct example shows how to specify that CA Spectrum should log an event, assign an event severity to it, and generate an alarm as a result of the event.

More information:

[Logging Event-Related Errors](#) (see page 25)

Add Comments in EventDisp Files

You can add comments to an EventDisp file using the # identifier. Any text that follows this identifier on a line is ignored when the EventDisp file is processed.

You must enter the # identifier as the first character on the line. Do not enter a space and then #, as this produces an error when the EventDisp file is processed. If the comment must span multiple lines, enter the # identifier as the first character on each line.

The following example shows proper usage:

```
# This is a valid comment.  
# This is a valid comment.  
0x3dc0004 E 50 A 2,0x3dc0001  
0x3dc0002 C 0x3dc0001
```

Appendix C: Event Format Files

This section contains the following topics:

[About Event Format Files](#) (see page 159)

[Location of Event Format Files](#) (see page 159)

[Contents of an Event Format File](#) (see page 160)

About Event Format Files

An *event format file* contains the message about the event that is displayed to users on the Events tab in OneClick when the event occurs. The message can contain references to the event variables that hold data retrieved from the variable bindings of the trap. There exists an event format file for each event generated by CA Spectrum.

The events provided with CA Spectrum all have event format files that define appropriate event messages. In addition, whenever you create a new, custom event (either using MIB Tools when you are mapping a trap to the new event or later using Event Configuration) the associated event format file is automatically created. This means that typically you should not need to manually create an event format file.

Important! It is recommended that you create and modify all event messages using Event Configuration, which updates the appropriate event format files on (only) the OneClick web server to which you are connected when you save the changes to a landscape. However, this appendix provides reference information on event format files if manual modifications are ever required.

Location of Event Format Files

The event format files that support the events provided with CA Spectrum are installed in the following folder:

```
<$SPECROOT>/SG-Support/CsEvFormat
```

Custom event format files that are created by MIB Tools or Event Configuration are installed in the following folder:

```
<$SPECROOT>/custom/Events/CsEvFormat
```

Each event format file is named Event<event_code>, where <event_code> is the 4-byte, hexadecimal event code assigned to the event. For example, an event with an event code of 0x12345678 has an event format file named Event12345678.

Contents of an Event Format File

If you modify an event format file manually, note the following:

- As you compose the event message, keep in mind that most of the information that a OneClick user receives about an event is via the message text that is associated with the event. For this reason, provide as much information about the event as possible in the message.
- An event message can consist of plain text and variables that reference specifics about the instance of the individual event. For information on the correct syntax for including variables, see [Variable Descriptions and Syntax](#) (see page 34).
- If there exists an event format file for an event, but no event map for the event exists in an event disposition file, the contents of the event format file are still displayed on the Events tab in OneClick when the event occurs.
- If no event format file exists for an event, a default message indicating this is displayed on the Events tab in OneClick.

Appendix D: Event Table Files

This section contains the following topics:

[About Event Table Files](#) (see page 161)

[Location of Event Table Files](#) (see page 161)

[Contents of an Event Table File](#) (see page 162)

About Event Table Files

An *event table file* does the following:

- Enumerates the possible values of a variable binding that is sent with a trap. The values can be attribute values in MIB tables, OID values, and integer bit values.
- Provides corresponding text values for the enumerated values.

When a trap is mapped to a CA Spectrum event, the variable bindings sent with the trap are mapped to event variables. This means that by referencing an event variable and the event table file for the appropriate variable binding you can define event messages that include the text values for the variable binding values. In turn, this means that the event message that is displayed to users in OneClick contains data that is specific to the trap that is sent.

You do not need to manually create event table files. All of the events provided with CA Spectrum that are mapped to traps that contain variable bindings with enumerated definitions in the MIB have supporting event table files. In addition, whenever you add trap support for a device that is not supported by default in CA Spectrum, and you map the traps to new, custom events using MIB Tools, an event table file is automatically created for each variable binding that meets this same criterion.

Note: This appendix provides reference information on the proper syntax for an event table file. However, typically you should not need to modify these files.

Location of Event Table Files

The event table files that support the events provided with CA Spectrum are installed in the following folder:

<\$SPECROOT>/SG-Support/CsEvFormat/EventTables

Custom event table files created by MIB Tools are installed in the following folder:

<\${SPECROOT}>/custom/Events/CsEvFormat/EventTable

Each event table file is named based on the associated device and variable binding.

Contents of an Event Table File

Event table files are used to enumerate the possible attribute values, OID values, or integer bit values in a variable binding sent with a trap, and to associate those values with corresponding text values that can be used in event messages.

Associate the Attribute Values in a MIB Table with Text Values

To associate an attribute value in a MIB table with a text value, the file must iterate each possible value in hexadecimal format and its associated text value, as shown in the following example:

```
0x00000001 Reconfiguration
0x00000002 Signal-Loss
0x00000003 Bit-Streaming
0x00000004 Contention-Streaming
0x000000ff None
```

Associate OID Values with Text Values

To associate an OID value with a text value, the file must iterate each possible OID value and its associated text value, as shown in the following example:

```
1.3.6.1.4.1.1563.1.2.1.1.3.2.36.2.6 dot6
1.3.6.1.4.1.1563.1.2.1.1.3.2.36.2.5 dot15
1.3.6.1.4.1.1563.1.2.1.1.3.2 dot7
```

Associate Integer Bit Values with Text Values

To associate an integer bit value with a text value, the file must iterate each possible integer bit value and its associated text value, as shown in the following example:

```
1 dsx1NoAlarm
2 dsx1RcvFarEndLOF
3 dsx1XmtFarEndLOF
4 dsx1RcvAIS
```

Appendix E: Probable Cause Files

This section contains the following topics:

[About Probable Cause Files](#) (see page 163)

[Location of Probable Cause Files](#) (see page 163)

[Contents of a Probable Cause File](#) (see page 164)

About Probable Cause Files

A *probable cause file* is an ASCII text file that defines the symptoms, probable causes, and recommended corrective actions for an alarm. When an alarm is generated as a result of an event, the text in the associated probable cause file is displayed on the Alarm Details tab in OneClick. In this way, OneClick users are provided with information that can assist in resolving the abnormal condition. There exists a probable cause file for each alarm generated due to a CA Spectrum event.

The events provided with CA Spectrum that generate alarms all have probable cause files that define appropriate alarm-related messages. In addition, whenever you create a new, custom event that generates an alarm (either using MIB Tools when you are mapping a trap to the new event or later using Event Configuration) the associated probable cause file for the alarm is automatically created. This means that typically you should not need to manually create a probable cause file.

Important! It is recommended that you create and modify all alarm-related messages using Event Configuration, which updates the appropriate probable cause files on (only) the OneClick web server to which you are connected when you save the changes to a landscape. However, this appendix provides reference information on the proper syntax to use if manual modifications are ever required.

Location of Probable Cause Files

The probable cause files that support the events and alarms provided with CA Spectrum are installed in the following folder:

```
<$SPECROOT>/SG-Support/CsPCause
```

Custom probable cause files that are created by MIB Tools or Event Configuration are installed in the following folder:

```
<$SPECROOT>/custom/Events/CsPCause
```

Each probable cause file is named Prob<alarm_cause_code>, where <alarm_cause_code> is an 8-digit (including leading zeros), hexadecimal code that identifies the probable cause of the alarm.

As a convention, events that generate alarms typically use their event codes as alarm cause codes.

Contents of a Probable Cause File

The contents of a probable cause file should include text only. On the first line, specify the cause of the alarm in all capital letters; this information is displayed as the alarm type (or title) on the Alarm Details tab. Also provide one section each on the following:

- The symptoms of the alarm
- The probable causes
- Recommended corrective actions

If there are multiple items under a category, enter the information in numbered list format.

Example: Probable Cause File Using Appropriate Syntax

UNKNOWN USER

SYMPTOMS:

The user's SMTP mail transaction failed with error code 550 - unknown user.

PROBABLE CAUSES:

- 1) The user may have entered an invalid SMTP mail login.
- 2) The SMTP server login account information may be incorrect.

RECOMMENDED ACTIONS:

- 1) Have the user check their username and try again.
- 2) If the username is correct, check the SMTP server login account information.

Glossary

alarm

An *alarm* is a CA Spectrum object that indicates that a user-actionable, abnormal condition exists in a model.

alert

An *alert* is an unsolicited message sent out by a managed element on a network. A more specific definition of an alert depends on the management protocol that is used to report the alert. In general, CA Spectrum uses SNMP as the management protocol to communicate with devices on a network.

event

An *event* is a CA Spectrum object that indicates that something significant has occurred within CA Spectrum itself or within the managed environment. Events always occur in relation to a model. When CA Spectrum receives an alert from a managed element on the network, in response, it generates a CA Spectrum event for the corresponding model if the received trap is mapped to an event.

event discriminators

Event discriminators are references to event variables that let you to generate alarms for events based on the values of the variables.

event format file

An *event format file* contains the message about the event that is displayed to users on the Events tab in OneClick when the event occurs.

event message

The *event message* is the message that is displayed on the Events tab in OneClick when the event occurs.

probable cause file

A *probable cause file* is an ASCII text file that defines the symptoms, probable causes, and recommended corrective actions for an alarm.

sequential window

A *sequential window* is a type of type time window in which non-overlapping time windows are examined, one after another, to determine if the requisite number of events has occurred within the time window. This type of time window is best suited for detecting a long, sustained train of events.

sliding window

A *sliding window* is a type of time window where if the specified number of events (or more) ever occurs within any window of the specified time period, the output event is created in response. This type of time window is best suited for accurately detecting a short burst of events.

trap

A *trap* is an *alert* that is generated by an SNMP-compliant device.

Index

A

- A flag • 119, 128
- about mapping to events • 105
- alarm severity mapping
 - creating • 53
 - defined • 52
 - modifying • 55
- alarm severity parameter • 119
- alarms
 - alarm cause codes • 47
 - defined • 11
 - generating using event discriminators • 50
 - persistent • 49
 - recommended actions • 49
 - severity levels • 45
 - severity mapping files • 52
 - unique • 49
 - user-clearable • 49
- AlertMap files • 14
 - alert codes • 107
 - comments • 109
 - defined • 103
 - error messages and • 110
 - location • 106
 - mapping variables • 110
 - OID map in • 108
 - processing alerts with • 106
 - syntax • 107
- alerts • 9

C

- C flag examples • 125, 127
- CA-authored events, modifying • 59
- columns, modifying • 25
- conditional alarm severity • 45
- conditional alarm severity mappings
 - creating • 53
 - defined • 52
 - modifying • 55

D

- D flag • 123
- developer IDs, obtaining • 13

E

- E flag • 118
- enable_event_variable_warnings parameter • 25
- errors in event disposition file syntax • 157
- event codes • 108
- event condition rules
 - configuring • 66
 - defined • 61
- Event Configuration
 - overview of the user interface • 17
 - starting from MIB Tools • 15
 - starting from OneClick • 15
- event counter rules
 - configuring • 80
 - defined • 64
- event discriminators • 121
- event disposition files
 - comments in • 158
 - defined • 115
 - location of • 116
 - synchronizing across landscapes • 20
 - syntax • 118
 - syntax errors in • 157
- event format files
 - contents • 160
 - defined • 159
 - location • 159
- event messages
 - defined • 33
 - event messages, example • 40
 - referencing attribute values in a MIB table • 37
 - referencing integer bit values • 39
 - referencing OIDs • 38
- event pair rules
 - configuring • 70
 - defined • 62
- event rate rules
 - configuring • 73
 - defined • 62
 - sequential window • 62
 - sliding window • 62
- event rules
 - creating • 65

- deleting • 93
- modifying • 92
- syntax • 129
- event series rules
 - configuring • 76
 - event series rules • 64
- event table files
 - attribute values in MIB tables • 162
 - contents • 162
 - defined • 161
 - integer bit values • 162
 - location • 161
 - OID values • 162
- event variables
 - copying to rule output events • 91
 - file syntax for copying • 151
- event_custom_override_warnings parameter • 25
- event_disp_default_log parameter • 25
- event_disp_error_file parameter • 25
- event_duplicate_action_warnings parameter • 25
- EventCombo rule syntax • 134
- EventCondition rule syntax • 135
- EventCounter rule syntax • 147
- EventPair rule syntax • 130
- EventRateCounter rule syntax • 132
- EventRateWindow rule syntax • 131
- EventRateWindowAttrParams rule syntax • 132
- events
 - clearing alarms with • 57
 - configuring • 32
 - creating from scratch • 30
 - defined • 10
 - error handling • 25
 - generate alarms from • 42
 - refreshing • 18
 - saving to landscapes • 18
 - searching for • 29
 - specify options for • 40
- EventSequence rule syntax • 133

F

- F flag • 123
- filtering events • 25

G

- generic trap identifier • 103

H

- Heartbeat rules
 - configuring • 83
 - defined • 64
 - syntax • 147

I

- initial alarm severity • 45

L

- landscapes • 20

M

- maintenance alarm severity • 45
- major alarm severity • 45
- MIB Tools • 9, 10, 14
- minor alarm severity • 45

N

- N flag • 122
- normal alarm severity • 45

P

- probable cause files
 - contents • 164
 - defined • 163
 - location of files • 163
- procedure_error_file parameter • 25

R

- R flag • 129
- Critical alarm severity level • 45

S

- sequential window • 62
- severity levels, alarm • 45
- Single Event Rules
 - configuring • 86
 - defined • 64
- sliding window • 62
- SNMPv2 InformRequest • 111
- SNMPv2 support • 111
- Solo Event Rules
 - defined • 65
 - Solo Event Rules, configuring • 89
- suppressed alarm severity • 45

T

T flag • 122

traps

defined • 9, 103

mapping to events • 14, 105

U

U flag • 121

user-defined event rule • 65

V

variable alarm severity • 45