HP OpenView Correlation Composer's Guide

For HP-UX, Solaris, Linux, Windows® 2000, Windows® XP, and Windows® 2003 operating systems

Manufacturing Part Number: T2490-90013

May 2004

Contents

Contact Information

Contacts

Please visit our HP OpenView web site at:

`http://openview.hp.com/`

There you will find contact information as well as details about the products and services HP OpenView has to offer.

Support

The "hp OpenView support" area of the HP OpenView web site includes:

- Downloadable documentation
- Troubleshooting information
- Patches and updates
- Problem reporting
- Training Information
- Support program information

1 Introduction

Scope

This document contains information you require to efficiently use the HP OpenView Correlation Composer. This guide contains information on:

- Understanding the Correlator Templates
- Using the Correlation Composer to define Correlators
- Use cases for using the Composer
- Troubleshooting the Composer during Runtime

Audience

This manual is intended for network personnel who maintain event correlation. Readers of this document are assumed to have the following background:

- General operational understanding of managed entities (network applications). In particular, an understanding of the event types generated by them.
- Familiarity with using GUI-based applications with mouse and menu-driven interface on UNIX workstations and Windows based machines.

On-line documentation

Product documentation is available in both hardcopy and browsable on-line format.

On-line documentation in Adobe Acrobat and/or postscript formats can be installed from the documentation CD-ROM.

To view documents from the external website, go to

http://ovweb.external.hp.com/lpe/doc_serv/

On-line Help

The HP OpenView Correlation Composer has an on-line help system that provides help on the functionality of the Correlation Composer.

To invoke the on-line help, click `Help->Overview` in the Composer's Main window.

2 HP OpenView Correlation Composer

This chapter introduces the HP OpenView Correlation Composer and introduces some of the commonly used terms. Additionally, it describes the Correlator Templates supported.

It describes the following terms and concepts in detail:

- "Correlator" on page 23
- "Correlator Store" on page 24
- "Correlator Templates" on page 25

Take time to read and understand these terms and concepts as they appear throughout the procedures described later in this manual.

HP OpenView Correlation Composer

The critical challenge for network and system operators today is dealing with the massive amount of information related to network, system and application problems. This information comes in many forms and the key is for your management solution to accurately understand what is important to present to your operators, what can be discarded, and what is needed for use by specialists who may need to follow a “bread-crumb trail” when diagnosing very complex problems.

The HP OpenView Correlation Composer (herein after referred to as Composer) is a graphical user interface used to parametrize and create problem specific Correlators. The problem specific Correlators uniquely identify a unit of logic to be applied to an event or set of events. The Composer enables users to tailor the event correlation behavior for Correlators, that are shipped with OpenView products, and simplifies the development of customer developed Correlators. Correlators can be used out of the box or can be easily fine tuned to fit your environment like a glove without any programming knowledge.

The Composer comes packaged with six Correlator Templates, that ease the creation of correlation solutions by providing correlation models for the most common correlation tasks. The Composer with the Correlator Templates forms the basis for simple correlation tuning and development.

The packaged Correlation Templates are:

- Enhance
- Multi-Source
- Rate
- Repeated
- Suppress
- Transient

In addition to the predefined Correlator Templates, that are explained in the sections below, custom requirements can be configured using the User-Defined Correlator Template.

Basic Concepts

Listed below are some of the definitions that you should be familiar with before defining Correlators.

Event

An event is an unsolicited notification, such as an SNMP trap or a CMIP notification generated by an agent process in a managed object or by a user action. Unsolicited notifications are referred so in the IP world.

Alarm

Event messages received from network elements are referred to as Alarms. Event messages are referred so in the Telecommunications world.

NOTE The terms Alarm and Event have been used interchangeably throughout this document.

Output an Event

When an event is output, it is visible to the end user. If the event is configured to be displayed, it will be listed in the Alarm Browser.

Discard an Event

When an event is discarded, it is not visible to the end user.

Event Type

The Event Type specifies the type of alarms that can be handled by the Composer. The Event types supported by the Composer are listed below:

Table 2-1 Supported Event Types

Event Type	Description
CMIP	CMIP based events
OVO	OVO messages
SNMP	Event traps
X733	X.733 based events

NOTE The Composer supports only one Event Type for a Correlator Store.

OVO supports only OpC messages and NNM supports only SNMP event traps.

CMIP and X733 Event Types are not supported at time of release.

Event Format

The network can receive events of different formats. The events can differ depending on the event type. The following sections describe the procedure to view events on the different platforms.

- **Network Node Manager**

  The NNM Alarm Browser actively notifies the operator when an important event occurs. You can browse through alarms to help you diagnose problems. The Alarm Categories window contains push buttons that correspond to each of the alarm categories. For more details on how to view the SNMP events, refer to *Managing Your Network with HP OpenView Network Node Manager*.

- **OpenView Operations**

  The Message Browser window displays all incoming messages from the OVO operator's managed environment. Double click on any message to display the Message Details window, listing full details of that particular message.

  For more details refer to the *HP OpenView Operations for UNIX Concepts Guide*.

- **ECS Engine Management (ecsmgr)**

  The ECS Engine can log events as they arrive at the engine. The event log file is an ASCII representation of the logged events in a form that can be displayed and edited using conventional text editing tools. To enable/disable logging of events that arrive at the engine, execute

  ```
  ecsmgr -log_events input on|off
  ```

  The input events log is written to `ecsin.evt0`. Output events log is written to `default_sout.evt0`.

  For more information on management of the ECS engine, refer to *HP OV ECS Administrator's Guide*.

Attributes

An alarm is a set of name value pairs where the name is referred to as an attribute. For example:

- In NNM, which handles SNMP traps, the attributes are `enterprise, agent-addr, specific-trap, variable bindings` to name a few
- In OVO, the attributes would be `MessageText, Application, Object` to name a few

Refer to Appendix C, "Event Attributes," on page 237 for a list of attributes for the standard Event Types.

The set of attributes that are visible within the Composer is configurable. To add Custom Message Attributes (CMA) for OVO or to add additional variable-bindings in NNM the configuration file should be edited. For more information refer to, "Define Event Attributes" on page 66.

Correlator

A Correlator uniquely identifies a unit of correlation logic to be applied to an event or a set of events. Every Correlator has three main sections:

1. **Alarm Definition section**

   The Alarm Definition section is divided into five subsections:

   - **Alarm Signature**

     The Alarm Signature (primary filter) forms the first level of filtering based on event attributes. Further processing takes place when an event matches all attributes set in the Alarm Signature. The Alarm Signature is a set of data structures constituting `Attribute Name`, `Operator` and `Value`

   - **Variables**

     Variables are names assigned to values. Once assigned, the name can be used in other sections of the Composer. There are two types of variables:

     — Global Constants - Global Constants are defined in the Global Constants section and can be accessed by any Correlator within the Correlator Store

     — Correlator Specific Variables - Correlator Specific Variables are defined in the Alarm Definition section of the Correlator and can be accessed ONLY within the scope of that Correlator

   - **Advanced Filter**

     This is an optional section. Alarms that have entered a Correlator can be further filtered based on the Advanced Filter Condition (secondary filter). This condition is typically used to define filters based on external factors like topology. (Contrast this to the Alarm Signature where the filter is defined purely on event attributes.)

- **Message Key**

  The Message Key identifies the instance of the Correlator under which the alarm is correlated, and is evaluated for each incoming alarm that passes the Alarm Signature and Advanced Filter. Alarms with identical Message Keys are correlated under the same instance of the Correlator.

- **Parameters**

  Parameters are specified to change the default behavior of the basic Correlator Template. Typically the time period for which the correlation is to be monitored is specified here.

2. **New Alarm Section**

   The user configures the specifications of new alarms to be created or altered in this section.

3. **Callback Section**

   A Correlator can result in events getting discarded or new events being created. There are two types of callback functions:

   - discard
   - create

   Everytime an event is discarded, the discard callback is invoked if configured. Everytime an event is either altered or created by a Correlator, the create callback is invoked if configured. This can be used to create an audit trail.

Correlator Store

The Correlator Store is an ASCII file that stores the configured Correlators. It is loaded at runtime to perform correlation.

Correlator Templates

The Correlation Composer supports some predefined Correlator Templates, which are explained in the sections below. Custom Requirements can be configured using the User-Defined Correlator Template.

Enhance Correlator Template

The Enhance Correlator Template can be used to:

- trigger the creation of one or more new alarms. Example - Create a new alarm that enumerates the set of customers affected by a failed entity.
- augment the information content of an alarm by modifying event attributes of an alarm. For example, modify the severity of an alarm depending on the customer who is affected.

By default, the alarm is enhanced only if no other Correlator has chosen to discard the alarm. However, this default behavior may be overridden.

NOTE

When an event is altered, a copy is made of the original event and then the copy is modified. Also, the attribute `unique_id` changes when an event is altered.

In the NNM environment

The UUID of the event changes when altered.

Multi-Source Correlator Template

The Multi-Source correlator Template is used to define a relationship between an arbitrary number of alarms, potentially from different sources that together form a logical set that identifies the problem. The set is considered complete if all alarms configured arrive within the specified time window.

Multi-Source Correlation can be used on set completion to:

- discard a subset of alarms
- modify a subset of alarms with attributes defined from any or all of the other alarms in the set
- create one or more new alarms with values called from attributes or pre-defined variables from the other alarms in the set

The set on completion can operate in two modes:

- Mode 1: The Multi-Source Correlator Template operates in this mode by default. When the set is deemed complete, the instance of the set remains in a completed state for the duration of the time window. This is typically used in a situation where all alarms from a source can be discarded if caused by the failure of another entity. The Multi-Source Correlator Template works in this mode when the `Set` button is NOT checked.

Example of Multi-Source correlation when operating in Mode1

When a BSC fails, all alarms emitted from the connected BTS can be discarded. The set in this example will have two alarms:

- The BSC alarm
- The BTS alarm (which is marked for Discard)

When a BSC alarm and a BTS alarm are received the set is complete. Subsequent BTS alarms are discarded until the time window expires.

- Mode 2: When Multi-Source correlation is configured to operate in this mode, it is expected that alarms will arrive in pre-defined sets. In this case the requirement is that the Correlator is applied as soon as the set is completed, after which the instance of the alarm is deleted.

Example of Multi-Source correlation when operating in Mode2

When the power to a MSC is lost - it emits a Mains_Fail, a Aircon_Fail and a Rectifier_Fail. The requirement is that if these alarms are received from the MSC within 10 seconds, then all three can be discarded and a new alarm must be created after set completion. Additionally, a subsequent alarm belonging to the set that arrives within the same window period, will need to wait for the other two before the second set is deemed complete. To illustrate, if a Mains_Fail arrives immediately after the first set is complete, the Mains_Fail will not be discarded till both the Aircon_Fail and Rectifier_Fail alarms arrive.

Rate Correlator Template

The Rate Correlator Template can be used to count the number of events occurring within a specified time period. If the count equals the value specified within the time period, the threshold is considered breached and a new alarm is created. The Correlator can be configured to:

- discard all alarms (regardless of rate) and emit only the newly created alarm when threshold is breached
- emit all alarms as they arrive and the newly created alarm (if any) when the threshold is breached

Repeated Correlator Template

The Repeated Correlator Template can operate in one of the following two modes:

- Mode 1: Duplicate alarms received within the window period of the first alarm are discarded. Repeated correlation operates in this mode by default.

  If the Correlator determines that the incoming alarm is to be discarded, the Correlator can also be optionally configured whether or not the alarm participates in other correlations before it is discarded.

  The user can also choose to send an update alarm at the end of the window period. This is typically used to create a new event indicating the number of alarms discarded by the first alarm in the window. Repeated correlation operates in this mode when the `Discard Duplicate` button is checked.

- Mode 2: Duplicate alarms are not discarded when the Correlator has been configured to operate in this mode. If there is a specification for a new alarm to be created, a new alarm is created for every incoming alarm. This mode is typically used to send a new alarm to replace the previously sent one, with the count of duplicate alarms received so far.

Suppress Correlator Template

The Suppress Correlator Template is used when a specific category of alarms needs to be discarded. Alarms that match all the conditions in both the Alarm Signatures and Advanced Filter will be discarded.

Transient Correlator Template

A transient failure is when the state of a managed entity changes to abnormal and then reverts to normal, in a short period of time. Transient Correlation is typically used to detect transient failures and when a transient failure is detected, associated events are discarded.

Additionally this model can be optionally used to monitor the rate of such transient failures and create a new alarm if a configured threshold is breached. (The threshold is considered breached if the number of transient pairs equals the configured breach value)

Example

Temperature_ON and Temperature_OFF alarms are generated when the temperature of a router exceeds the threshold or falls below the threshold. Transient correlation can be used to discard both the Temperature_ON and Temperature_OFF alarms if the Temperature_OFF alarm is received within 5 minutes of occurrence of the Temperature_ON alarm.

User-Defined Correlator Template

The User-Defined Correlator Template is used to implement a requirement when none of the other correlation models, either by itself or in a combination, can meet the correlation requirement.

Alarms that meet the conditions specified in the Alarm Signature and Advanced Filter will invoke the Input Function specified.

The Input function can be of any type, 'C', Perl or the built-in type. The return value of the Input function determines the action to be taken on the alarm, which could be to create new alarm, discard the alarm, hold the alarm for a specified period and so on.

If the Input function requested that the alarm be held, then after the specified period the output function is invoked. The return value of the Output function determines the action to be taken on the alarm, which could be to create a new alarm(s), discard the alarm, output the alarm and so on.

Modes of Correlation Composer

The Composer is designed to operate in two modes, namely:

- **Developer's Mode**

  The Composer Developer can set up, create or modify correlation logic for the network environment. The Developer is responsible for configuring the Composer, setting up Operator access rights using Security files and deciding the area of operation for the Operator using Namespace files.

  Refer to Chapter 5, “Correlation Composer for the Developer,” on page 119 for more information.

- **Operator's Mode**

  The Composer's Operator maintains correlation in the network. Each operator handles a part of the correlation logic and is responsible for this continuous maintenance. The Operator can refine the correlation logic based on the network requirements. The Operator has limited access to the Composer. The access is governed by the permissions set by the Developer in the Security file and the area of operation specified in the Namespace files.

  So, while assigning a user's role as Operator, appropriate conditions and permissions must be set in order to enable the Operator to efficiently manage the network. To simplify this task, Security files and Namespace files are provided in order to link the Operator to that portion of the network. In this manner, many operators can be given access to different portions of the network by linking them to Security and Namespace files.

Correlator Template Evaluation Precedence

Each Correlator is implemented by a discrete decision-making mechanism based on the Correlator Template used. If the filters of two Correlators are defined such that they admit the same alarm, then both the Correlators are applied to the alarm. When an event participates in multiple Correlators, the following rules are applied to determine the outcome:

- The order of Correlator evaluation is Suppress followed by Repeated, all other Correlators(in parallel) and finally Enhance correlation.
- If the Suppress and Repeated Correlators choose to discard the alarm, the user can optionally choose to allow the event to participate in other Correlators before it is discarded.
- The Enhance Correlation is run last. The event is enhanced if and only if no other Correlator decides to discard that event except when '`Enhance Always`' is enabled in the Enhance correlator Template.
- An event is output *if and only if* no other Correlator has discarded it.

3 Using the Correlation Composer

This chapter provides an overview and guide to using the Correlation Composer.

NOTE All descriptions in this chapter are relative to the Composer having been started in the Developer’s mode.

Getting Started

A typical configuration of the Correlator Store involves the following:

- The first step is to clearly and fully understand the correlation requirement. It would be helpful if this requirement is documented in terms of distinct steps.

  As part of the above exercise also document the set of alarms that participates in the correlation. For example, if the requirement is to discard node_down alarms from routers, then document how to recognize the node_alarm from the router. This involves recognizing a node_down alarm(the attributes of the alarm to be examined to determine if the alarm is a node_alarm) and then further examine the attributes to be examined to determine if the node_down alarm is from a router.

  The first half, that is, recognizing a node_alarm, typically maps to the Alarm Signature section. The second half, that is, recognizing a router alarm would involve the Advanced Filter if there is no attribute within the alarm to indicate that this alarm is from a router.

- The next step is to map the requirement to one or more of the packaged Correlator Templates (including User-Defined). Refer to Chapter 8, “Use Cases,” on page 159 for examples where multiple Correlator Templates are used to meet a requirement.

Software Prerequisites

You need the following software to install and run the Composer.

- You need to be running one of the following operating systems:
  - HP-UX version 11.0, 11.11, 11.22 PA, or 11.22IA
  - RedHat Advanced Server 2.1
  - Solaris 2.8 or 2.9
  - Windows 2000, Windows XP, or Windows 2003
- Java 1.4 or above must be installed on the machine where Composer will be run.

Start the Composer

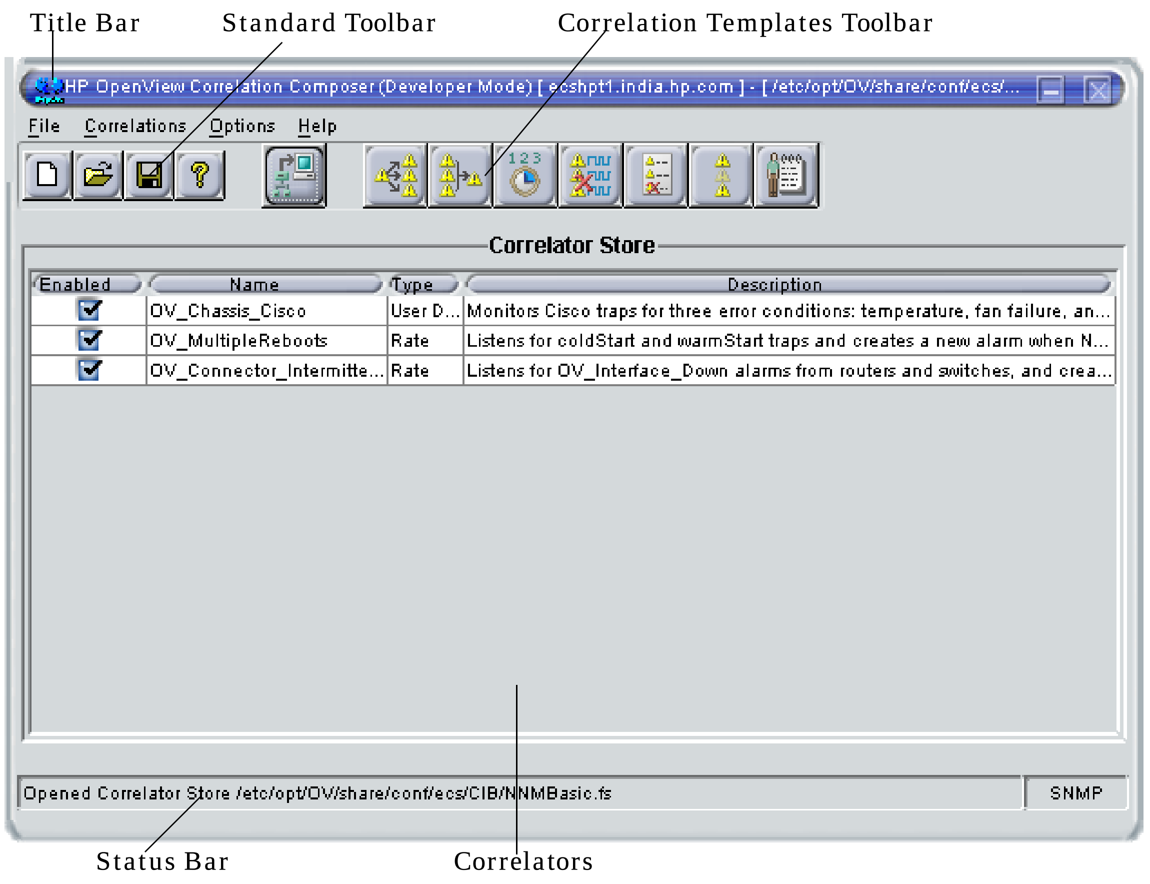
The method of starting the Composer depends on the environment in which you want to invoke it. The following tables describe the various methods of invoking the Composer.

Table 3-1 **Starting the Correlation Composer**

Environment	Method
HP OpenView NNM	In the ECS Configuration Management GUI, select the row with `Composer` and select the `[Modify]` button.
Command Line	• **`ovcomposer -m o`** to start in Operator mode • **`ovcomposer -m d`** to start in Developer mode. Refer to the *ovcomposer* manpage for more details.
Windows	1. Click on the `[Start]` button and point to `Programs`. 2. Select `<program_group>->Correlation Composer`

When started the Correlation Composer window is displayed. Refer to the figure below to see what the Composer looks like when a Correlator Store is open.

Figure 3-1 The HP OpenView Correlation Composer


Title Bar
Standard Toolbar
Correlation Templates Toolbar
HP OpenView Correlation Composer (Developer Mode) [ecshpt1.india.hp.com] - [/etc/opt/OV/share/conf/ecs/...
File
Correlations
Options
Help
Correlator Store
SNMP
Status Bar
Correlators

Enabled	Name	Type	Description
✓	OV_Chassis_Cisco	User D...	Monitors Cisco traps for three error conditions: temperature, fan failure, an...
✓	OV_MultipleReboots	Rate	Listens for coldStart and warmStart traps and creates a new alarm when N...
✓	OV_Connector_Intermitte...	Rate	Listens for OV_Interface_Down alarms from routers and switches, and crea...

Opened Correlator Store /etc/opt/OV/share/conf/ecs/CIB/NNMBasic.fs

The opening panel consists of the standard menus and options required to define Correlators. The standard tool bar consists of some of the most frequently used menu options.

Help System

To display the online help for the Composer, press F1. To display the Help Table of Contents, select `Help:Table of Contents` from the menu.

Mouse

To select an item, position the mouse pointer over the item and click the left mouse button.

Click the right mouse button to pop up a menu of the frequently used options.

Localized Information

Correlator Stores can be opened independent of the locale in which they were created. Correlator Stores developed in one locale can be opened, modified and used seamlessly from another locale. User descriptions in the Correlators written using English locale are visible in all the other locales even if descriptions are not localized in the respective locale. This helps the users to read and localize the descriptions, if they know both English and the current locale language. Since the logic part is kept separate from the language part, logic changes are seamless across locales while the user description changes are private to each locale.

Exit the Composer

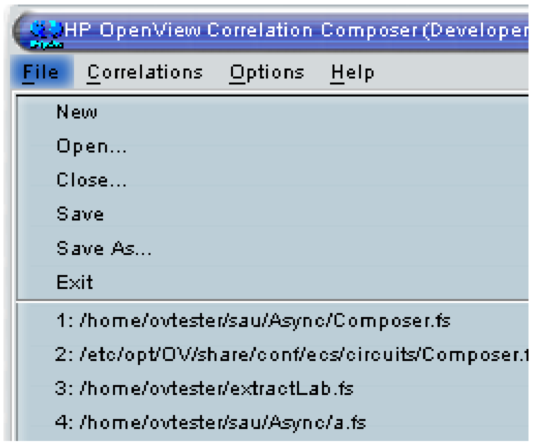
To close the Correlator Store file, click `File:Close`. This closes only the Correlator Store that is currently being configured.

To exit from the Composer session click, `File:Exit`.

Menu Commands

The following sections describe the active menu commands in the Composer.

File Menu

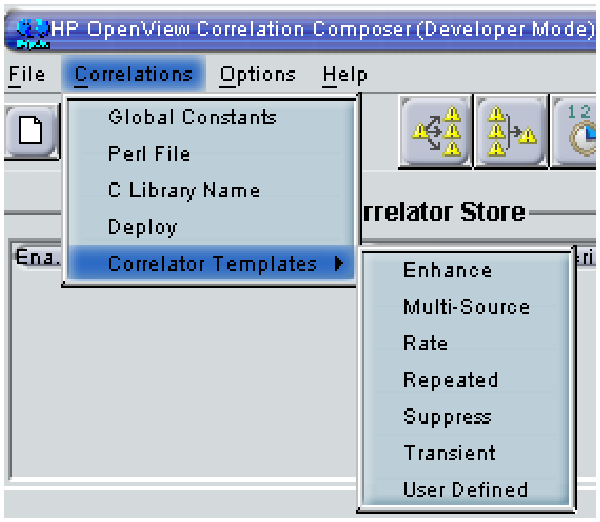

HP OpenView Correlation Composer (Developer
File
Correlations
Options
Help
New
Open...
Close...
Save
Save As...
Exit
1: /home/ovtester/sau/Async/Composer.fs
2: /etc/opt/OV/share/conf/ecs/circuits/Composer.f
3: /home/ovtester/extractLab.fs
4: /home/ovtester/sau/Async/a.fs

Menu Item Name	Description
`New`	Creates an empty Correlator Store. If an existing Correlator Store file is still open, Composer closes this file. If you have not yet saved the file, it prompts you to save it.
`Open`	Displays the file browser to enable you to find and open a previously saved Correlator Store.
`Close`	Closes the opened Correlator Store. If you have not yet saved the file, Composer prompts you to save it.
`Save`	Saves the Correlator Store. This item is disabled if the configuration file has not been modified since the last save.
`Save As`	Saves the Correlator Store to a different name. Selecting this, displays the file browser in which you enter the new file name.
`Exit`	Closes the Correlator Store and exits the Composer. If the current configuration file has unsaved changes, the Composer prompts you to save the file before exiting.

In the NNM environment

Only Save and Exit options are available in the NNM environment. NNM provides a set of built-in Correlator Stores to enable maintainance of Correlators specific to that environment. All updates must be made to these files. For more details on the Composer in the NNM environment refer to Chapter 10, “Correlation Composer for NNM,” on page 205.

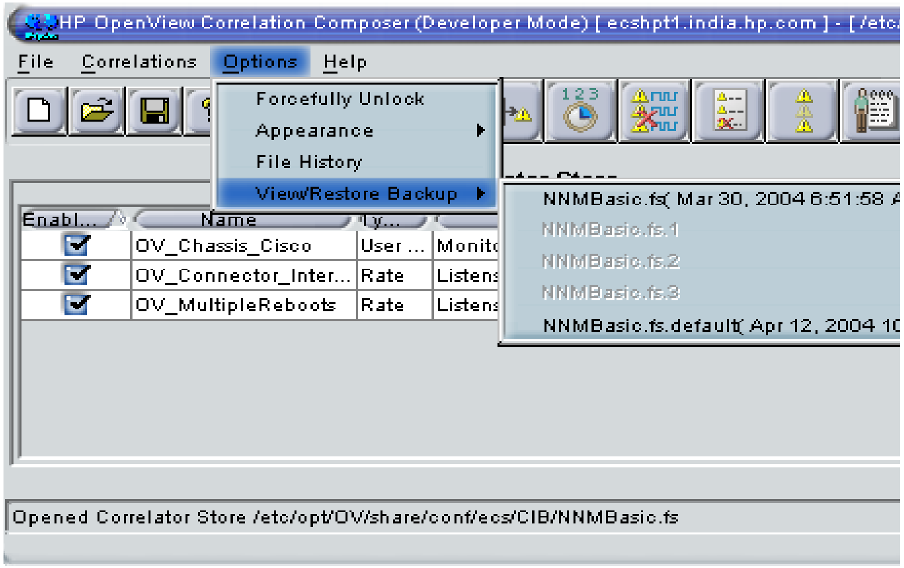
Correlations Menu



Menu Item Name	Description
Global Constants	Displays the Global Constant Definition window. This window enables you to enter the constants that are global across the Correlator Store.
Perl File	Displays the Perl File Name window where you specify the Perl file that contains the Callback function.
C Library Name	Displays the C Library Name window where you specify the C library name.
Deploy	Deploys the Correlator Store files to the ECS engine. This option in enabled only when the Composer is started in the Operator mode.

Menu Item Name	Description
`Correlator Templates`	Displays a submenu for selecting the kind of correlation.

Options Menu


HP OpenView Correlation Composer (Developer Mode) [ecshpt1.india.hp.com] - [/etc
File
Correlations
Options
Help
Forcefully Unlock
Appearance
File History
View/Restore Backup
NNMBasic.fs(Mar 30, 2004 6:51:58
NNMBasic.fs.1
NNMBasic.fs.2
NNMBasic.fs.3
NNMBasic.fs.default(Apr 12, 2004 10
Enabl...
Name
Ty...

Enabl...	Name	Ty...	
✓	OV_Chassis_Cisco	User ...	Monit
✓	OV_Connector_Inter...	Rate	Listen
✓	OV_MultipleReboots	Rate	Listen

Opened Correlator Store /etc/opt/OV/share/conf/ecs/CIB/NNMBasic.fs

Menu Item	Description
`Forcefully Unlock`	Provides mutually exclusive access to the Correlator Store. For more information “Mutual Exclusive Access to Correlator Store files” on page 141.
`Appearance`	Displays a submenu for selecting the kind of Look and Feel of the interface.
`File History`	Displays the File History window that displays the list of recently opened files.
`View/Restore Backup`	Displays a submenu to select the version of backed up file. For more details refer to “View Backup Files” on page 67.

Viewing Backed Up Correlator Store files

The Composer provides the ability to recover from a disaster by taking regular backups of the Correlator Store files.

The Correlator Store file when created and saved the first time or opened the first time, creates a default (if it does not exist already) backup file which remains constant throughout the life of the Correlator Store. Changes made to the Correlator Store file and when saved the first time in the current session (involves the time between opening of the file and closure of file) will form the contents of a backup file. The backed up file is identified by the extension `'.1'` appended to the filename at the time of Save. Consecutive first saves in future sessions results in the creation of renewed backed up files identified with extensions `.2, .3` and so on. Backed up files roll from `.1->.2->.3` and so on. Backed Up files can be edited by the user and group only. The tables below list the file permissions for the Correlator Store and the Backed Up files.

Table 3-2 File permissions for Correlator Store and Backed Up files

	HP-UX, Solaris and Linux		**Windows**	
	Correlator Store	**Backed Up Files**	**Correlator Store**	**Backed Up Files**
User	read+write	read+write	read+write	read+write
Group	read+write	read+write	read+write	read+write
Others	read	read	read+write	read+write

The number of backed up files that can be taken is configurable by the user. By default, three backups will be taken for every Correlator Store file. However, this number can be overridden by editing the configuration file CO.conf. Refer to “Define Event Attributes” on page 66 for more details.

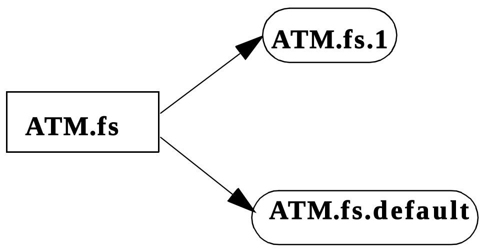
Example 3-1 Example of Rollback to Backed up files

Consider the Correlator Store file ATM.fs and assuming that the number of backups taken is configured to 3.

1. A back up file ATM.fs.default is created when the Correlator Store is created and saved or opened the first time. This file remains constant throughout the life of ATM.fs.

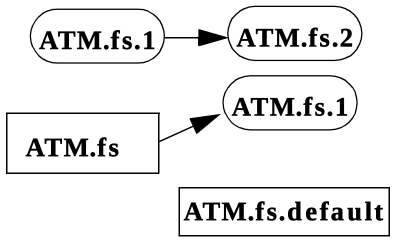
2. Further edits made to the file and saved will constitute the file ATM.fs. Note that, all the latest changes are reflected in this file.
3. Opening a new session (with ATM.fs) and saving it the first time in the current session results in the movement of the existing ATM.fs to ATM.fs.1 before writing the new changes to ATM.fs.

Session 1

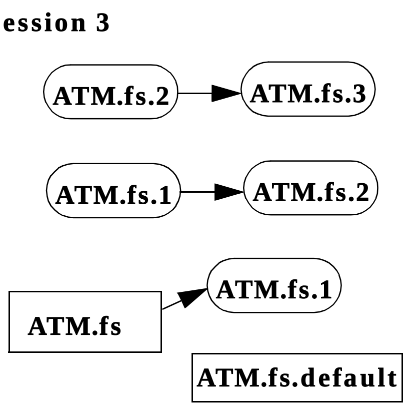


4. Further edits made to the file and saved will constitute ATM.fs.
5. Opening a new session (with ATM.fs) and saving it the first time in the current session results in movement of
   a. ATM.fs.1 to ATM.fs.2
   b. existing ATM.fs to ATMS.fs.1

Session 2



6. Opening a new session (with ATM.fs) and saving it the first time in the current session results in the movement of
   a. ATM.fs.2 to ATM.fs.3
   b. ATM.fs.1 to ATM.fs.2
   c. previous ATM.fs to ATM.fs.1
7. Edits made to this file and saved will constitute ATM.fs.



8. Opening a new session (with ATM.fs) and saving it the first time in the current session results in rolling of the backup files to the next version number.

   Hence, the backed up file ATM.fs.3 (from session 3) is now updated with new contents.

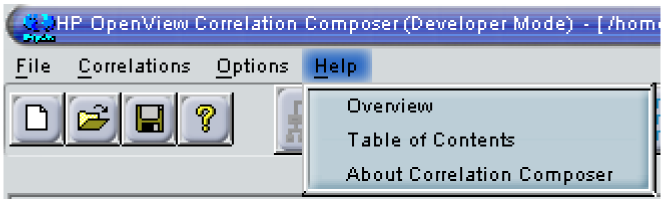
To open a backed up file, select `Options->View Backup-><`*`Version of file`*`>`. Selecting this option displays the following message:

```
You are now viewing an archive version of the file. To
restore this version, select the Save button.
```

This message warns the user of viewing an older (backed up) version of the file. If you want to make this backed file the latest file, save the file to revert changes to the latest file. When you revert to the latest file, consecutive backed up files roll down to accommodate new changes.

IMPORTANT It is recommended that the decision to revert changes to the latest file be made judiciously, since there is always a threat that data can be overwritten erroneously.

Help Menu

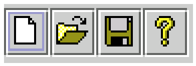

HP OpenView Correlation Composer (Developer Mode) - [/hom
File
Correlations
Options
Help
Overview
Table of Contents
About Correlation Composer

Menu Item	Description
`Overview`	Displays the Composer Online Help.
`Table of Contents`	Displays the Composer Online Help Table of Contents. You can also view the Help Index from this window.
`About`	Displays the current release and copyright information for the Composer and associated software.

Toolbar buttons

Standard Toolbar

The standard toolbar buttons provide shortcuts to frequently used menu commands.



The standard toolbar buttons and their functions are listed below:

Table 3-3 **Standard Toolbar Buttons**

Toolbar Button	Description
`New`	Creates a new Correlator Store. Displays the Input Event Type window to allow you to select the Input Event Type.

Table 3-3 **Standard Toolbar Buttons (Continued)**

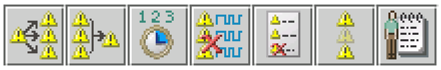
Toolbar Button	Description
`Open`	Displays a file browser to enable you to find and open a previously saved Correlator Store.
`Save`	Saves the current Correlator Store. This item is disabled if the Correlator Store has not been modified since the last save.
`Help`	Displays the Help contents of the Correlation Composer.

In the NNM environment

Only `Save` and `Help` buttons are available in the NNM environment. NNM provides a set of built-in Correlator Stores to enable maintainance of Correlators specific to that environment. All updates must be made to these files. For more details on the Composer in the NNM environment, refer to Chapter 10, "Correlation Composer for NNM," on page 205.

Correlator Templates Toolbar

The Correlator Templates Toolbar provides shortcuts to the various Correlator Templates.



The Correlator Templates and a brief description are listed below:

Table 3-4 **Correlator Templates Buttons**

Toolbar Button	Description
Enhance	Displays the Enhance Correlation window to define parameters to add more information to an alarm before the alarm is output.
`Multi Source`	Displays the Multi Source Correlation window to define events of different kinds and output events with enriched information.

Table 3-4 Correlator Templates Buttons (Continued)

Toolbar Button	Description
`Rate`	Displays the Rate Correlation window to define parameters to maintain a count of event arrival and output a new event based on this count.
Repeated	Displays the Repeated Correlation window to define parameters to discard events of similar type.
`Suppress`	Displays the Suppress Correlation window to define parameters to discard a certain class of events.
`Transient`	Displays the Transient Correlation window to define parameters and correlate based on some threshold values.
`User Defined`	Displays the User Defined Correlation window. The users can define Correlators based on their requirements.

Deploy Button

The Deploy button provides a shortcut to the Deploy the Correlator Stores to the ECS engine.

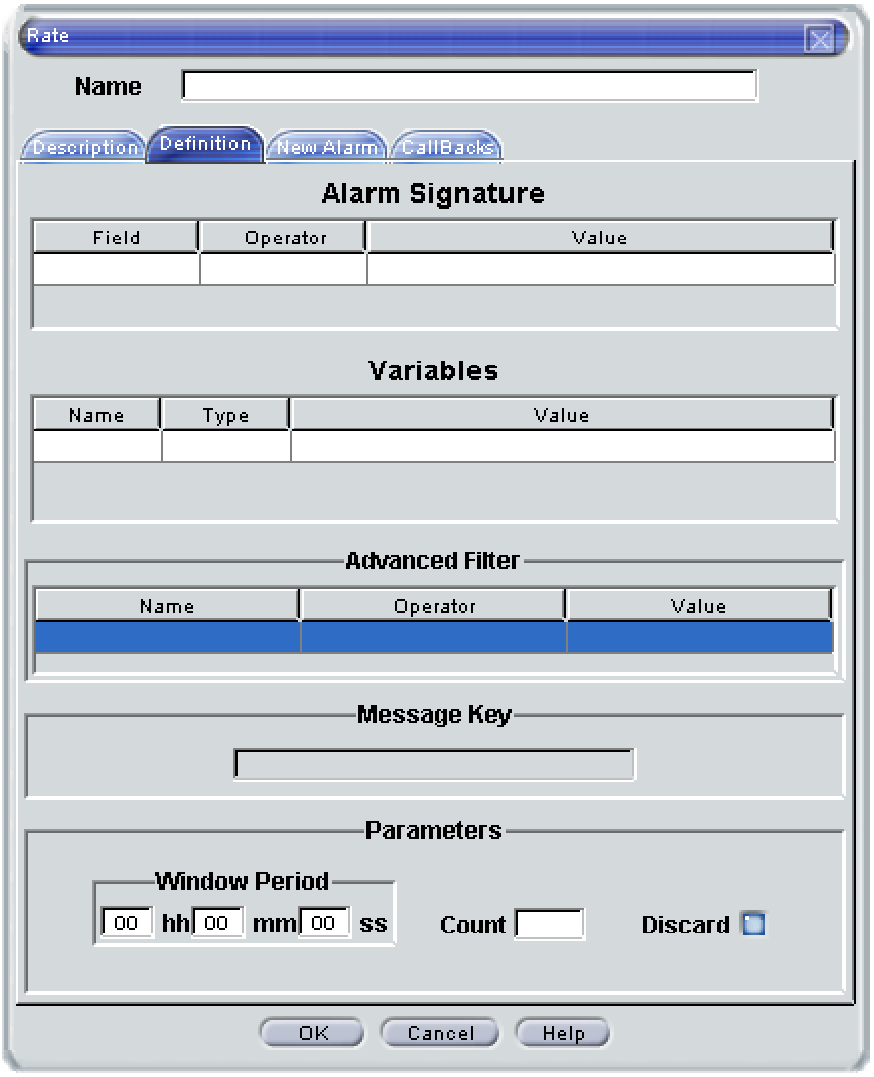
Table 3-5 Deploy Button

Toolbar Button	Description
`Deploy`	Merges the Correlator Stores and loads the merged Correlator Store into the ECS engine. For more details on the Deploy procedure refer to “Deploying the Correlator Store” on page 142.

Correlator Window

A Correlator is defined and parameterized from the Correlator configuration window. The following figure represents a typical Correlator configuration window.

Figure 3-2 Correlator Window



The Correlator configuration is primarily divided into three sections:

1. Alarm Definition
2. New Alarm Creation
3. Callback Functions

In addition to the above three panels, a panel is provided to describe the Correlator. The Correlator opens with the Description panel open.

1. Alarm Definition

The first section in the Correlator configuration is the Alarm Definition. The Alarm Definition is provided from the `Definition` panel in the Correlator window. The Alarm Definition consists of the following subsections described below.

- **Alarm Signature**

  The Alarm Signature(primary filter) is the first level of filtering performed by the Composer. Alarms whose attributes match the attribute specification specified in the Alarm Signature will be processed further. The Alarm Signature is a set of tuples constituting `Attribute` name, `Operator` and `Value`.

  The standard Event Attributes are listed in Appendix C, “Event Attributes,” on page 237. Event Attributes vary depending on the Event Type selected. To customize the list of attributes, refer to the “Define Event Attributes” on page 66.

  The `Operator` matches the `Attribute` type to the `Value` entered. Supported Operators are listed below

Table 3-6 Valid Operators

Operator	Description	Value must be one of
=	Equals	Integer, Float, String or Object ID
!=	Not Equal	Integer, Float, String or Object ID
<	Less than	Integer or Float
>	Greater than	Integer or Float

Table 3-6 Valid Operators (Continued)

Operator	Description	Value must be one of
`<=`	Less than or equals	Integer or Float
`>=`	Greater than or equals	Integer or Float
matches	The pattern specified by value is matched against the value contained in the attribute	Integer, Float, String or Object ID
`does NOT match`	The pattern specified by value is NOT present in the attribute	Integer, Float, String or Object ID
`is in list`	Equals the list of values returned by the attributes	List[a]
`is NOT in list`	Is not equal to any of the values specified in the list	List

a. All values must be enclosed within square brackets([ ])

matches - If the pattern represented by Value is present in the attribute. For regular pattern matching expressions refer to Appendix D, “Pattern Matching,” on page 249.

Example - “1234510”(which is the value contained in the specified attribute) `matches` “10”. Here the string pattern “10” is searched for in the string “1234510”.

Example - `“1234510” matches “^10”` returns False.

If the value extracted from the attribute is not a string, then the value is converted to a string and the pattern is matched.

Example - If the attribute enterprise contains an Object ID, 1.2.3.4.5.6 and the requirement is to discard traps with an enterprise Object ID of 1.2.3.4, then the following expression would meet the above requirement

`enterprise matches 1.2.3.4.`

Internally 1.2.3.4 is converted to a string “1.2.3.4” and this pattern is looked for in the string version of the enterprise. This however would also match an enterprise id of 5.6.1.2.3.4 which is not the requirement. The following ensures that the correct pattern is matched:

```
enterprise matches “^1.2.3.4”
```

Note that the pattern is given as a string.

does NOT match - The string pattern represented by Value is not present in the attribute. The `does NOT match` operator is the opposite of the `match` operator.

Example - The Value “1020” must not be present in the attribute.

is in list - The value entered must be present in the list of values

Example - a `is in list` [a, b, c, d]

is NOT in list - The value entered must not be present in the list of values entered.

In the NNM environment

— The attribute `agent-addr` in the SNMP trap is represented as a string in the `‘.’` notation. For example, if the agent-addr is passed to a function, it will be passed as “a.b.c.d” and NOT a.b.c.d. If the agent-addr needs to be set while creating/altering an alarm the variable carrying the agent-addr should also be in the same format. For example, to set the agent-addr to an IP address of 15.10.76.143, define a variable whose value is a string like “15.10.76.143”.

— variable bindings are a name value pair where the name is always an Object ID. While specifying a new alarm that has variable bindings both the name and the value for each variable binding needs to be specified. variable bindings start with an index of 0.

Automatic Variables

Apart from the standard Event Attributes and the user-defined variables, there are some automatic variables maintained by the Composer. The Automatic Variables maintained by the Composer are

— **AlarmCnt -** The AlarmCnt attribute maintains the count of the number of alarms that entered the Correlator. For example, if the correlation being used is Rate correlation, the attribute AlarmCnt maintains the count of the number of alarms arriving.

  The AlarmCnt variable is accessible while creating new alarms and defining Callback functions in the New Alarms and Callbacks sections respectively.

— **CorrelationDuration -** Correlation Duration is the actual time taken for the Correlator to be applied. For example, while using Rate correlation, a new alarm can be triggered when the rate exceeds 5 in 30 minutes. But if the rate has been breached in the 10th minute, then the CorrelationDuration has the value 10 bound to it.

- **Variables**

  Variables are names given to values to be used while defining Correlators or Global Constants. All attributes in the alarms can be accessed as variables where the variable name is the attribute name.

Table 3-7 Variable Types

Type	Description
`Constant`	Value entered by the user is bound to the variable name
`Extract`	Matched string is assigned to a variable
`Function`	Represents data returned from a function
`Combine`	Combination of two or more variables
Lookup	Represents data returned from a datastore lookup

Constant - Constant values are used for reference while defining a Correlator. The variable name is bound to the value specified in the value field.

Example: A variable `ErrStr` is bound to the value “`Temperature High. Check for A/C Failure`”. The variable `ErrStr` can be used locally within the Correlator under which the variable is declared.

Extract - Sub strings within the event attribute can be extracted.

Example:

`<*.err_text> - <#.link_num> on Signalling Set <#.set_num>`

matches a message such as

`Link Failure - 10 on Signalling Set 2`

and assigns `Link Failure` to `err_text`, `10` to `link_num`, 2 to `set_num`.

Refer to “Pattern-Matching” on page 251 for more examples on pattern matching.

Function - The return value of a function can be bound to a name. Functions can be called synchronously or asynchronously. Refer to “Writing External Functions to be called as the Input/Output functions of a User-Defined correlation” on page 112.

Example - Assume a variable `X` which is bound to the return value of a function, say `xyz`. Also assume that `xyz` returns an integer. In this case, the variable `X` would be bound to the integer value returned by the function `xyz`.

In cases where functions return more than a single value, individual elements can be accessed via the Built-In function `getByIndex`. For example, lets say a function, `myFunction`, returns 10, 20, 30 and this is bound to a variable, say `myVariable`. To access individual elements use the Built-in function `getByIndex`. For more details on Built-In functions, refer to Chapter , “Composer Built-in Functions,” on page 147.

Combine - Variables can be combined to form a new variable by using the Combine operator.

Example: The Combine operator concatenates variables to a single string.

Assume the following variables defined with values as mentioned

- `a constant ‘Hello’`
- `b constant ‘World’`
- `c constant ‘Rate is’`
- `d constant 10`
- `e constant 20`

`Combine [a b]` results in `“Hello World”`

`Combine [c <`*`AlarmCnt`*`>]` results in `"Rate is <`*`AlarmCnt`*`>`

NOTE When as integer and a string are combined the resultant output is a string.

`Combine[10, 20]` results in results in `1020`

Lookup - Values from a Data Store[1] can be queried and bound to variables using the Lookup operator. Parameters for the `Lookup` are one or more variables. The values referred to by these variables are concatenated and the resulting value is used as the key in the Lookup to the datastore.

The format of the Data Store is

`ADD DATA(keyValue, ReturnValue)`

where,

`keyValue` must be an integer or string, and

`ReturnValue` can be any datatype

The Data Store file can have multiple such lines. A comment begins with two hyphens(--). The first line in the file must be the header whose format is

`#path#date#version#0`

Example - Assume the Data Store loaded, has one entry `ADD DATA("Overheated", 80)`. Also assume a variable X whose value is "Overheated". If X is used as a parameter to Lookup, the value returned will be 80.

Using two variables Y and Z whose values are "Over", "heated" would result in a key value of "Overheated" and the same value 80 would be returned.

1. The Data Store contains a table of global values. Unique keys identify each table entry these keys are used to select a value from the table.

TIP

Typically, the Data Store is used to hold static topological information. (One could think of using scripts to run once a day, say midnight, to create the datastore file and update the ECS engine with the newly created file).

Rules for variable evaluation

By default, a variable is evaluated(via a function) when it is used. To override this default behaviour, the user is given the choice to select the point at which the variable can be evaluated. Following are the different times at which the variable can be evaluated

— Default

The default behaviour of variables. The variable is evaluated when it is used.

— Event In

The variable is evaluated when the event participates in the Correlator after having passed the primary and secondary filter conditions.

— Correlator Creation

The variable is evaluated when the Correlator is instantiated.

For example, in Repeated Correlation, the variable is evaluated when the Correlator is instantiated.

— Correlator Deletion

The variable is evaluated when the instance of the Correlator is deleted.

Refer to the Table 3-8, “Correlator Creation and Deletion,” on page 57. Note that all parameters other than the standard attributes displayed in the pop up menu SHOULD be previously defined as variables.

IMPORTANT

A variable is evaluated ONLY once. For example, if a variable has been flagged to be evaluated at EventIn but the variable is used in the Advance Filter, then the variable gets evaluated when the Advanced Filter is processed and is *not* re-evaluated when the event enters the Correlator.

Functions flagged for evaluation at EventIn, Correlator Creation and Correlator Deletion will ALWAYS be invoked synchronously.

An Asychronous function whose parameters have not yet been evaluated at the point at which the function has been invoked, cannot depend on a parameter that is evaluated through another Asychronous function.

- **Advanced Filter**

  Events into a Correlator can be further filtered based on the Advanced Filter(secondary filter) condition. The Advanced Filter is a set of data structures constituting `Name, Operator, Value.`

  Refer to “Alarm Signature” for Attributes and Operators supported.

Example of Advanced Filter

The requirement is to generate a new alarm if a Router_Failure alarm is received from a Core router.

Just by examining the Event Attributes, it is not possible to deduce if the alarm is emitted from a Core router. To solve this problem, define a variable, say `isCoreRouter`. Let this variable be bound to the return value of a function-`GetIsRouter`. This function takes the `agent-address` as its parameter and returns 1 if the router is a Core router, else returns 0. In the Advanced Filter, define a Correlator that checks if the variable, `isCoreRouter` is set to 1 which ensures that the Correlator is applied only to Core routers.

- **Message Key**

  The Correlator defined is merely a template to indicate the interaction of alarms. For example, consider a Multi-Source Correlator that has two alarms defined - a `router_down` and a `interface_down` alarm. The Correlator is configured such that the `router_down` alarm suppresses individual `interface_down` alarms emitted from the same router. The idea here is that if a router fails, then discard individual interface(component) failures from the same router. The Correlator specified, however is generic in the sense that it applies to all `router_down` and `interface_down` alarms. It is obvious that an `interface_down` alarm should be suppressed only if the router to which this interface belongs has also failed. The mechanism to tie alarms together, in this case, the router down alarm to the interface alarm from the same router, is the Message Key. The Message Key is evaluated when the alarm enters the Correlator. Alarms that evaluate to the same value of the Message Key are correlated together. Taking the above example, the name of the router could be used as a Message Key assuming that the router name can be extracted from both the router and the interface down alarms. The Message Key could be a physical entity like an interface, a router OR a logical entity like a service, customer, PVC.

When an alarm enters the Correlator (the event has passed both the primary and secondary filters for the Correlator), the Message Key is evaluated. If there does not exist an instance of the Correlator with the evaluated Message Key, an instance of the Correlator is created with the Message Key. This is referred to as Correlator Creation. On the other hand, when the event comes in, if there exists an instance of the Correlator for the same Message Key then the incoming alarm is correlated under the Correlator for this Message Key. In other words this alarm will be correlated with the other alarm(s) that evaluated to the same Message Key. The Message Key is necessary only when two or more alarms need to be related. For example, in Suppress there is no requirement for a Message Key as the Correlator will be applied to all alarms that meet the Alarm Signature and Advanced Filter conditions.

The point at which an instance of the Correlator is deleted(referred to as Correlator Deletion) is dictated by the semantics of the correlation model.

Table 3-8 Correlator Creation and Deletion

Correlation Model	Message Key Required?	Operational mode (if any)	Point of Correlator creation	Point of Correlator deletion
Enhance	No		NA	NA
Multi-Source	Yes	Mode 1	When the first alarm with a Message Key for which no Correlator has been instantiated	• When the time window expires from the time the last event for this Message Key entered the system

Table 3-8 Correlator Creation and Deletion (Continued)

Correlation Model	Message Key Required?	Operational mode (if any)	Point of Correlator creation	Point of Correlator deletion
		Mode 2	When the first alarm with a Message Key for which no Correlator has been instantiated	• When the set is complete • When the set is not complete - Time Window after the last alarm for this Message Key entered the system
Rate	Yes		When the first alarm with a Message Key for which no Correlator has been instantiated	• At the point where the rate is breached • When the time window expires from the time the last event for this Message Key entered the system
Repeated	Yes		When the first alarm with a Message Key for which no Correlator has been instantiated	Time window after Correlator instantiation
Suppress	No		NA	NA

Table 3-8 Correlator Creation and Deletion (Continued)

Correlation Model	Message Key Required?	Operational mode (if any)	Point of Correlator creation	Point of Correlator deletion
Transient	Yes	No threshold specified	When the first Fail alarm with a Message Key for which no Correlator has been instantiated	• When a Clear alarm comes within the window period of the Fail(immediately on getting a pair) • When the window period expires without a Clear alarm coming(no pair detected)
		Threshold specified		• When threshold number of pairs is received in the Time Window(at the point of threshold being breached) • Threshold not breached and the threshold period after the last pair was created
User-Defined	NA	NA	NA	NA

Examples of Message Keys

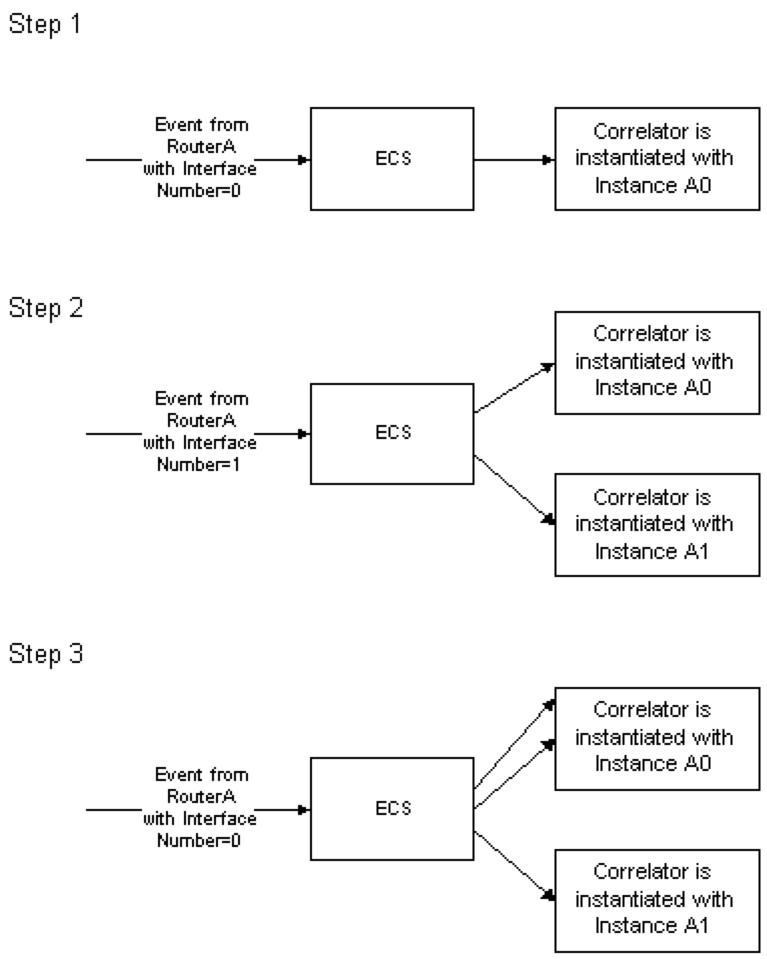
— The requirement is to trigger a new alarm indicating that the alarm rate is too high, when the number of alarms received from the same router crosses 20 within 1 hour.

All events matching the Alarm Signature, are categorized as Router Alarms. But there has to be a mechanism by which to examine Router alarms emitted from the same router. To solve this problem, assign the `agent-address` as the Message Key.

— Monitor the rate at which interfaces on a router are failing.

We have configured a Correlator called IF_Rate where the Message Key = x+Interface Number (say varbind0), where x can be some attribute. Refer to the figure below to understand the example.

Figure 3-3 Message Key


Step 1
Event from RouterA with Interface Number=0
ECS
Correlator is instantiated with Instance A0
Step 2
Event from RouterA with Interface Number=1
ECS
Correlator is instantiated with Instance A0
Correlator is instantiated with Instance A1
Step 3
Event from RouterA with Interface Number=0
ECS
Correlator is instantiated with Instance A0
Correlator is instantiated with Instance A1

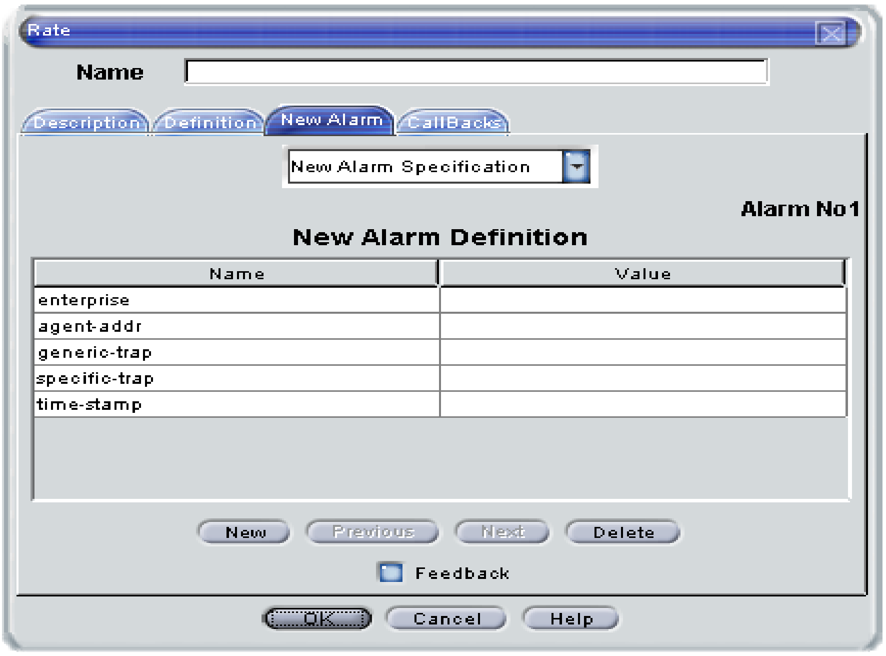
- Parameters

  Parameters are set to change the default behavior of the basic Correlator type. The functionality of the parameters can vary across the different Correlator Templates. The functionality of these parameters is discussed in Chapter 8, "Use Cases".

2. New Alarms Creation

The second section in the Correlator configuration is the New Alarm Creation section. The New Alarm Creation specification is provided from the New Alarm panel in the Correlator window. Correlators can include definitions to trigger new alarms with information content that is useful for the operator. For example, in the case of Repeated Correlation, a new alarm can be output at the end of the window period, reporting the number of alarms that arrived in this time period.

Figure 3-4 **New Alarms section**



Name	Value
enterprise	
agent-addr	
generic-trap	
specific-trap	
time-stamp	

New alarms can be created in two ways:

- **Alter Specification**

  New alarms can be created by altering some of the attributes of the existing alarm. The Alarm Specification is a tuple that constitutes `Field, Mode` and `Value`.

  Field - It is the field that has to be altered. The Field is a drop down menu and displays all attributes for the selected Event type. Refer to Chapter C, “Event Attributes,” on page 237 for Event attributes.

  Mode - There are two modes that can be used to alter the event’s attributes.

  — Replace

    The new value replaces the event attribute of the original alarm.

  — Append

    The new value specified is appended to the existing event attribute.

  Value - The value of event’s attributes is appended or replaced with new values.

NOTE The `Alter Specification tab` will not be enabled if the Correlation type being defined is Multi-Source.

The attributes displayed to alter an alarm, are always the attributes of the last alarm that arrived. For example, while using Transient correlation, the attributes displayed will always be that of the last Clear Alarm.

- **New Alarm Specification**

  New alarms can be created with all new attributes. The New Alarm Specification is a tuple that constitutes `Name` and `Value`. The New Alarm Specification displays all the mandatory attributes that have to be entered to create a new alarm.

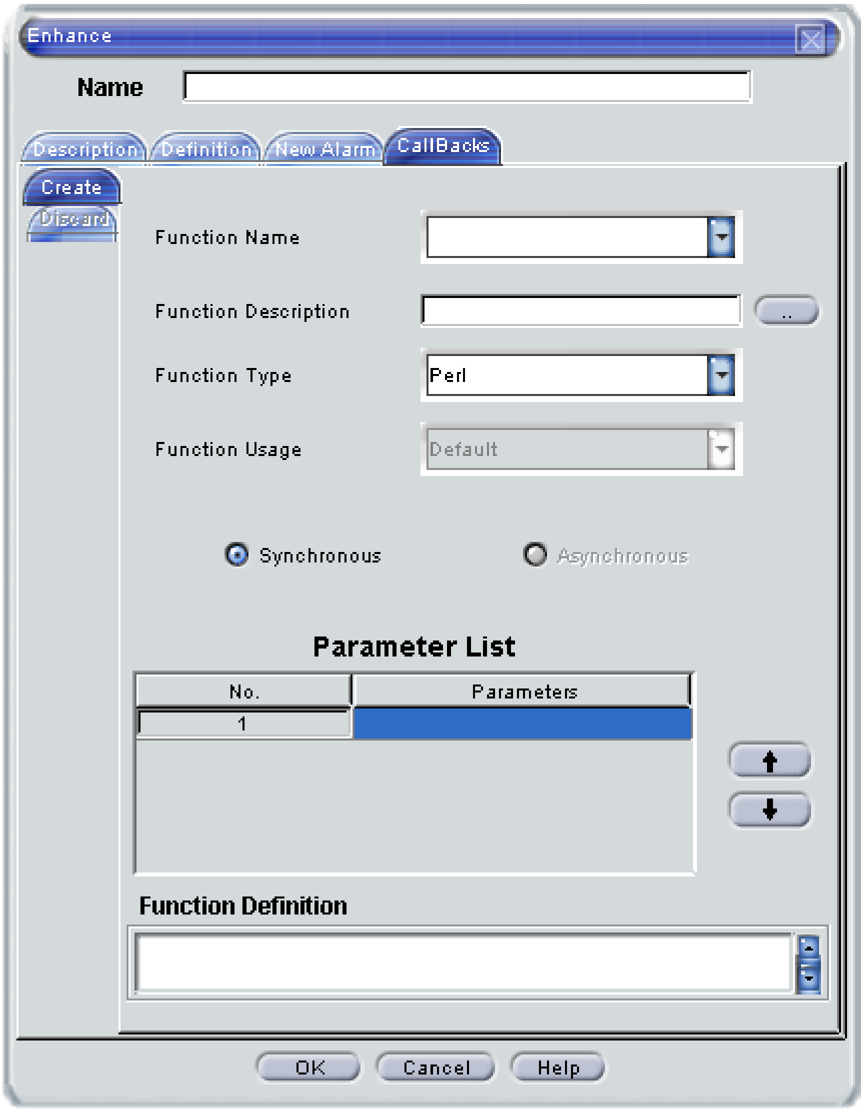
Additionally, the user is given the option to feedback the alarm into the circuit. Select the `Feedback` button if the new event has to be fedback into the circuit and participate in other Correlators.

NOTE The New Alarm tab is not available for Suppress Correlation, as there are no new events that can be output.

3. Callback Functions

The third section in the Correlator configuration is the Callback Functions section. The Callback Functions is provided from the `CallBacks` panel in the Correlator window. An alarm can be discarded or output after it has participated in correlation. Similarly, when a new alarm is created, the user can optionally choose to invoke an external function to perform user-specific functions like logging or issuing a trouble ticket. The external function can be written in C, Perl or built-in functions can be used and the parameters for the function are selected from the Parameter List table. The location of the C or the Perl file containing the external function is specified from the Composer. Callback functions can be defined from the `Callbacks` panel of the Correlator window.

Figure 3-5 Callback Function



Callback functions can be called at two instances:

- Create - The Callback function(if defined) is called when a new alarm is created.
- Discard - The Callback function(if defined) is called when the alarm is discarded.

The Create/Discard tabs are used to define when the Callback function will be called.

NOTE The Create and Discard Callback functions can be called only Synchronously.

For information on how to write Perl and C functions, refer to "Writing External functions in C" on page 97 and "Writing External functions in Perl" on page 105.

Define Event Attributes

The configuration shipped with the Composer contains the Standard Event Attributes as defined by the Event Type(SNMP, OpC). The user can configure additional attributes by editing the configuration file `CO.conf` placed under the default directory where the Composer is installed. The table below lists some examples

Operating System	**Composer Installed directory**	**Configuration file directory**
HP-UX	`/opt/OV/bin`	`/opt/OV/bin`
Linux	`/opt/OV/bin`	`/opt/OV/bin`
Solaris	`/opt/OV/bin`	`/opt/OV/bin`
Windows	`C:\OpenView\bin`	`C:\OpenView\bin`

NOTE `Root` access is required to make any changes to the configuration file.

Why add additional attributes?

In NNM

The default `CO.conf` file contains `variable bindings` from 0 to 12. If additional `variable bindings` need to be added, edit the `CO.conf` file. To edit the configuration file:

1. Open the file `CO.conf` in any standard text editor.
2. As an example. Search for the second occurrence of the string `"SNMP"` and add the string for the varBind[13] in the same format.

   To add the variable varBind[13] to the existing list, add `varBind[13]->name` and `varBind[13]->value` in two separate lines.

   Note that for each variable bindings, two entries need to be added - one for the name and one for the value.
3. Save the file and close the file.

In OVO

Assume that a Custom Message Attribute(CMA) called Customer-ID needs to be added.

1. Open the file `CO.conf` in any standard text editor.
2. As an example. Search for the second occurrence of the string“`OpC`”.

   Add `Customer-ID` in a new line.
3. Save the file and close the file.

View Backup Files

Composer provides an option to view backed up files. The number of backed up files visible by the user is by default 3. This number can be overridden by editing the field MAX_BACKUP in the configuration file, CO.conf.

To edit the file,

- Replace the default number (3) under the heading MAX_BACKUP by the number of backups files you would like to view.

The maximum number of backups that can taken for a Correlator Store is 20.

NOTE

The Composer must be restarted after any change is made to the configuration file `CO.conf`.

4 Developing Correlators with Composer

This chapter provides a structured approach to creating a Correlator Store file and contains:

- "Planning the configuration" on page 71
- "Correlator Store" on page 73
- "Correlators" on page 79

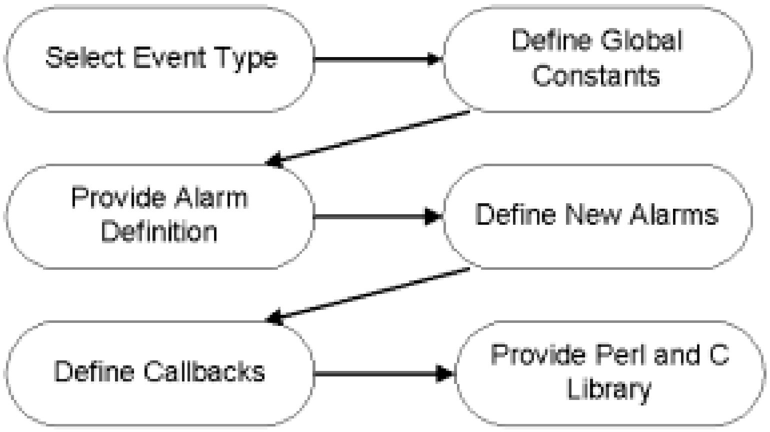
Planning the configuration

A typical configuration of the Correlator Store involves defining the various Correlation Templates, parameters and alarms required to be provided as input to the circuit. To ensure a smooth configuration, planning to cover the following is essential

Step 1	Define Event Type: for which the Correlator Store is being created. Refer to “Step 1: Event Type” on page 76
Step 2	Define Global Constants: It is a set of values that is defined globally for a set of Correlators. Refer to “Step 2: Define Global Constants” on page 77
Step 3	Provide Alarm Definition: Create and define the alarms to be correlated. Refer to “Step 3: Define Alarm Definition” on page 79
Step 4	Define New Alarm: Define the new alarm. Refer to “Step 4: Define New Alarms” on page 87
Step 5	Define Callback Functions: Provide Callback function definitions. Refer to “Step 5: Callback Functions” on page 91
Step 6	Load Perl and C library: Provide Perl and C library location. Refer to “Step 6: Load Perl and C Library” on page 93

NOTE Step 2, Step 4 and Step 5 and optional. However it is recommended that you follow the procedure as explained if you have to define any of the parameters explained in the steps.

Figure 4-1 Planning the Configuration



The sections that follow describe the procedure to configure the Correlator Store. It is recommended that you read and understand the rest of this chapter to become familiar with the Correlation Composer.

For information on the menu items, dialog boxes and windows refer to Chapter 3, “Using the Correlation Composer,” on page 31.

Correlator Store

A Correlator Store contains a set of Correlators which define correlation requirements for the network.

Create and Save the Correlator Store

To create a new Correlator Store:

- Select `File:New` from the menu.
- Click on the New icon in the Standard Toolbar

If a file is already open, the Composer prompts you to save the file. Note that the default Event Type is SNMP. To change the Event Type refer to “Step 1: Event Type” on page 76.

To save the Correlator Store file:

1. Select `File:Save` from the menu to display the file browser.
2. Enter the Correlator Store file name in the `File` panel.
3. Select `[OK]` to save the file.

File Naming Restrictions

Use file names that start with a letter and contain only letters, digits and underscore(_). For example: `my_configuration` is a valid file name. The extension `.fs` is supplied automatically to the filename entered.

Location of Correlator Store

You can save a Correlator Store under any directory. Ensure that the correct path is specified before saving the file.

Opening an existing Correlator Store

To view a Correlator Store file:

1. To select the filename that you want to open

- Click File:Open and select the Correlation Store name from the file browser window.
- Click on the Open icon in the Standard Toolbar.

2. The Open file browser window is displayed. Select the name of the file you want to open.
3. Click `[Open]`. The Correlator Store file is displayed.

IMPORTANT Correlator Stores created in a Composer version prior to Version 3.3 must be migrated to the latest vesion. To migrate Correlator Stores refer to “Migrate existing Correlator Stores” on page 75.

Modify an existing Correlator Store

Once you have created a Correlator Store, you can modify its properties whenever required. To modify the Correlator Store:

1. Select `File->Open` to open the Correlator Store. This opens the file browser window.
2. Select the filename from the file browser window. The Correlator Store with the Correlators is displayed.
3. To open a Correlator, you can do one of the following
    - Select the Correlator in the table and double click the mouse button. The Correlator window opens.
    - Select the Correlator in the table and right click the mouse button. From the menu displayed select `Modify`. The Correlator window opens.
4. Make the required changes as you did when you created the file.
5. To save the changes, select `File->Save`. The file is saved with the same name.

To save the changes made into the different file, select `File->Save As`.

In the NNM Environment

Only `Save` and `Exit` options are available. NNM has a default Correlator Store and all updates must be made to this file.

Migrate existing Correlator Stores

Correlator Stores created using Composer prior to Version 3.3 *must* be migrated to the latest version. The `csmigrate` script residing in the directory `$OV_CONTRIB/ecs` migrates Correlator Stores created prior to version 3.3 to the latest version. To migrate to the lastest version, type

```
csmigrate.ovpl <Correlator Store name> -lang
<ENGLISH|JAPANESE|CHINESE> -o <final Correlator Store name>
```

where,

<*Correlator Store name*> is the name of the Correlator Store that must be migrated

<ENGLISH|JAPANESE> is the native language of the Correlator Store that must be migrated

<*final Correlator Store name*> is the name of the Correlator Store after migration.

Step 1: Event Type

The Event Type decides the kind of events that will enter ECS. The event types supported by the Composer are listed below

Table 4-1 Event Types supported

Event Type	Description
`CMIP`	CMIP based events
`OVO`	OVO messages
`SNMP`	SNMP traps
`X733`	X.733 based events

The Event Type is selected at the time of creating the Correlator Store.

To select the Event Type

1. Select `File->New`. The `Input Event Type` window is displayed
2. Select the Event Type from the list and click on `[OK]`.

NOTE The default Event Type is SNMP.

The Correlator Store can be created for one Event Type at a time. If you want to change the Event Type, close the currently opened Correlator Store and repeat Steps 1 and 2.

In the NNM Environment The Correlation Composer in the NNM environment does not allow the Event Type to be changed.

Step 2: Define Global Constants

Values can be bound to names and referred to by this name while defining Correlators.

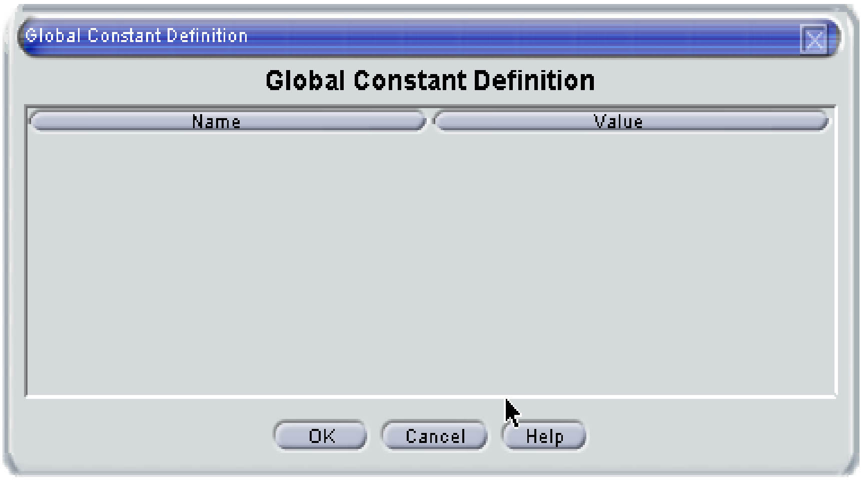
Global Constants

The parameters that you enter to define the named value pairs can refer to the global constants. For example, a named value called `DEVICE` that you have defined can have the event attribute “`eventInfo.notificationIdentifier`”.

To define the Global Constants

1. Select `Correlations->Global Constants` from the Main Menu. The `Global Constants Definition` window is displayed.

Figure 4-2 Global Constants



2. Enter the following data:

   - `Name` - this will be the name with which the constant will be referred to.
   - `Value` - this is the value for the Name.

These values can be referenced anywhere inside the Correlator Store. The table below lists the value types

Table 4-2 Value types for Global Constants

Value Types for global constants	Example
Integer	123
Float	123.45
String	“1234” or ‘abcd’
OID	1.2.3.4

3. To add more Global Constants, right click the mouse button and select `[Add]`. A new row is added to the Global Constants table.
4. When you have finished defining Global Constants, click on `[OK]` to close the window.

TIP To delete any Global Constant, you can do one of the following

- select the Global Constant, right click the mouse button and select `[Delete]`
- select the Global Constant, and hit the `[Delete]` key

Step 3: Define Alarm Definition

This step describes the procedure to define the Correlators.

Correlators

Define a new Correlator

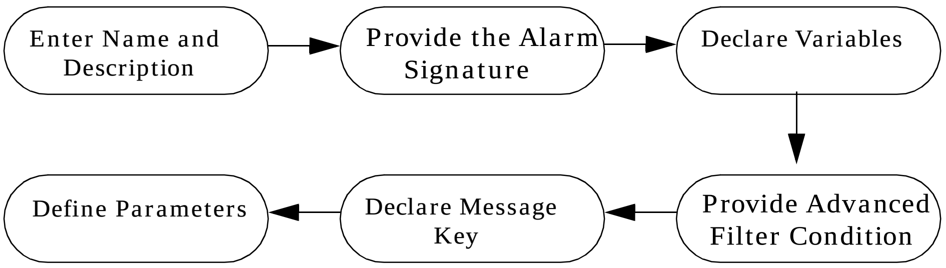
Every Correlator has a Name that identifies it uniquely, Correlation Description that states briefly what the correlation is expected to do and the Correlator definition itself. Refer to “Defining the Correlator” on page 80 on how to define a Correlator.

To create a new Correlator

- Select `Correlation:Correlation Templates`. From the menu displayed select the Correlator Template you would like to create.

The Correlator window is displayed. The flowchart below summarizes the main tasks to define a Correlator.

Figure 4-3 **Task flow to create a Correlator**


Enter Name and Description
Provide the Alarm Signature
Declare Variables
Provide Advanced Filter Condition
Declare Message Key
Define Parameters

Defining the Correlator

The following section describes the procedure to define a Correlator.

Follow the procedure given below to create and define a Correlator:

1. In the Correlator window enter the Correlator Name in the `Name` text box. Note that the alarm will be referred by this name throughout the Correlator Store.

   Naming Restrictions

   Use names that start with a letter and contain only letters, digits and underscore(_). Usage of special characters like!, @, #, !, ^, & and * is not allowed. For easier reference let the name indicate the problem type. For example “`Generator_OFF`” is a valid name.

2. Enter the `Description` of the alarm. The description can very briefly state what the cause of the alarm can be and what the Correlator is expected to do.

3. **Alarm Signature**

   Click on the `Definition` tab. The Alarm Definition panel is displayed. Define the Alarm Signature. Alarm Signature is a set of values that specifies a filter. Select the `Field` name from the drop-down list.

   Table C-1 lists the valid Event Attributes for the various Event Types supported by the Composer.

4. Select the `Operator` value from the drop-down list. For a list of valid operators refer to Table 3-4 on page 43

5. Enter the value of the field for which the Alarm Signature is described. To enter a value, click in the value cell and type in the value.

   To select a Global Constant, double click in the Value cell. A pop up menu displaying the previously defined constants is displayed. Choose the constant from the menu displayed.

6. **Variables**

   Declare the Variables. Variables are names with associated values that can be used inside the Correlator definition. Enter a name to the variable.

7. Select the variable type from the drop down menu. For a list of variable types supported refer to Table 3-5 on page 46.
8. Depending on the Variable type chosen, the value has to be entered. Click inside the Value cell to enter the value. For a detailed procedure on how to enter values for the different Variable types refer to “Defining Variable Types” on page 81.
9. **Message Key**

   Define the Message Key. To enter the Message Key, click in the Message Key box. A pop up menu displays all possible values. Select the variable/attribute that you would like to declare as Message Key.

NOTE There is no Message Key required to be defined for the Suppress, Enhance and User-Defined Correlations.

10. **Parameters**

    Enter the parameters. Refer to Chapter 8, "Use Cases," for a description of all the buttons in the parameters section.
11. **Advanced Filter**

    Define the Advanced Filter(if any). Select the Attributes, Operator and the Value from the pop up menu displayed in the respective columns. Note that this is a non-editable field. All values to be displayed must be previously defined.
12. Click `[OK]` to complete the creation of the Correlator.

Defining Variable Types

The following sections provide procedures to assign values to Variables.

Constants Constant values are used for reference while defining a Correlator. To define constant values:

1. Click in the Value cell and enter the value. The value is displayed in the cell.

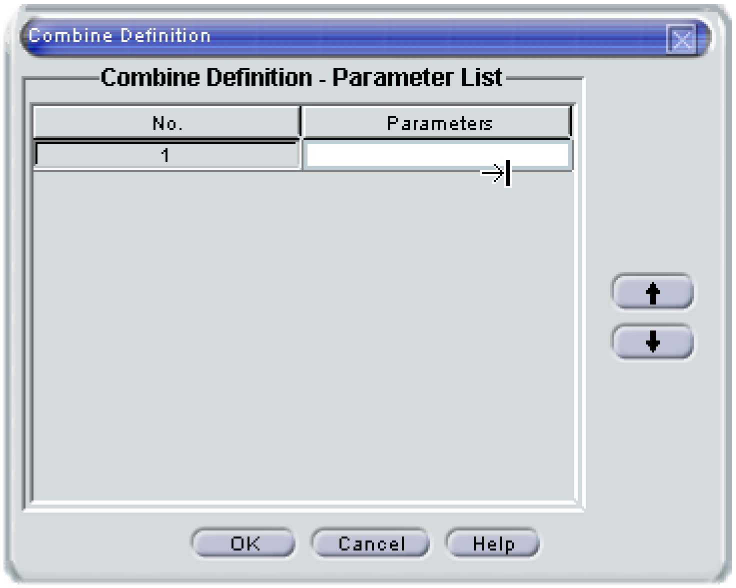
While defining variables in NNM environment

Specification of Object IDs do not have the leading dot. For example, 1.2.3.4 is valid while.1.2.3.4 is not.

Combine A new variable can be defined, by combining two or more previously defined variables, attributes or Global Constants. Follow the procedure given below to combine values of two or more variables.

1. Click in the value cell. The Combine Definition window is displayed.

Figure 4-4 Combine Definition



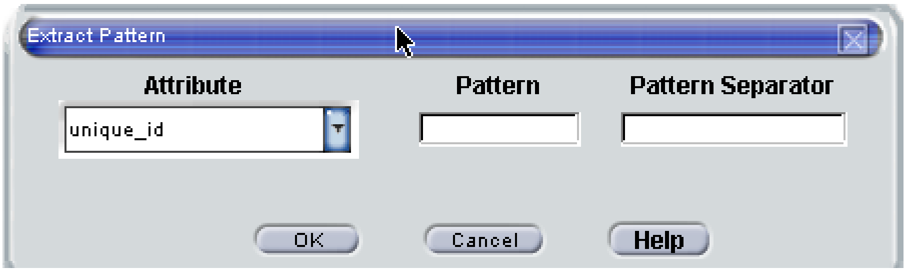
2. The Parameter list has to be filled to combine the variables. Click on the Parameters cell. A pop up menu displaying all attributes, pre-defined variables and Global Constants is displayed.
3. Select the attribute or variables from the pop up menu. The selected item is displayed in the parameters cell.
4. To add a variable or attribute to the list, right click the mouse button and select Add. A new row is added. A pop up menu displaying all attributes and variables is displayed when you click in the Parameters cell. Select the variable/attribute/Global Constants that you want to combine.
5. Repeat Step 4 to combine more variables/attributes.

Lookup A variable can represent data returned from a Datastore lookup. The procedure is the same as when variables are to be combined.

Extract Pattern Event attribute values can be extracted and used inside correlation definition. Follow the procedure described below to define an extract pattern.

1. Select Extract from the drop down menu. Click in the value cell. The Extract Pattern window is displayed.

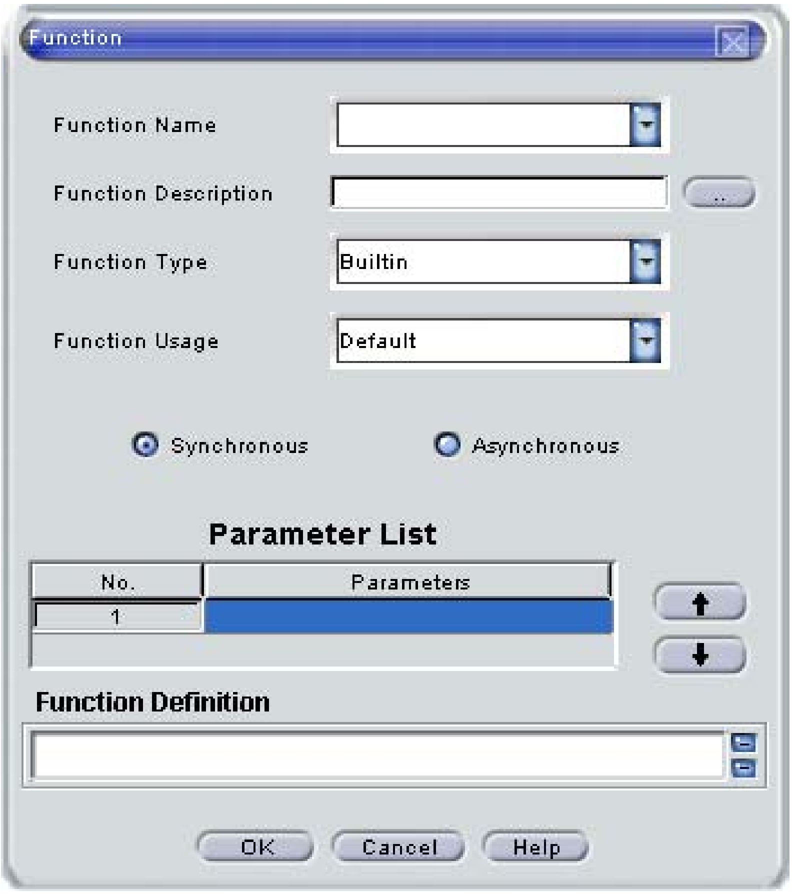
Figure 4-5 Extract Pattern



2. Select the attribute from which you want to extract a substring.
3. Enter the pattern in the pattern text box. Refer to Appendix D, “Pattern Matching,” on page 249 for extract pattern examples.
4. Enter the pattern separator in the Pattern separator text field. The Pattern Separator is by default an empty space.
5. Click on [OK] to close the Extract Pattern window.

Function The variable type can be a function whose return value is bound to the name of the Variable.

Figure 4-6 Function Definition window



Function

Function Name:

Function Description:

Function Type: Builtin

Function Usage: Default

[✓ Synchronous] Asynchronous

Parameter List

No.	Parameters
1	

Function Definition

For information on how to write functions in Perl and C, refer to “Writing External functions in Perl” on page 105 and “Writing External functions in C” on page 97. To define the function

1. Select the `Function Type` from the drop down menu. The function types are

   - C function
   - Perl function
   - Built-in function

Depending on the selection, the appropriate function names are loaded in the `Function Name` field. To enable loading of C or Perl function names, refer to “Configuring UserDevelopedFuncDetails.xml File” on page 103.

2. Select or enter the `Function Name`. This is the name of the function to be called.
3. Provide a brief description of the function in the `Description` tab. To view the entire description window click on the `[...]` button.

   While Perl and C functions are external functions to be supplied by the user, built-in functions are packaged with the Composer.

NOTE You can also specify the library name while defining the function. The function name can be prefixed with the C library name, where this function resides. For example, if a function named `BSCName()`resides in a library called `SNMPlib`, you could mention function name as `SNMPlib:BSCName()` in the `Function Name` text box.

To refer to a function named `function1` in a Perl file `User1.pm`, enter `User1::function1`. Refer to “Support for Multiple Perl files” on page 107.

4. Select the phase at which the external function has to be invoked. The function can be invoked at the following phases:
   - Default - The external function(written in Perl or C) is called.
   - Event In - The function is called when the event enters ECS.
   - Correlator Creation - The function is called when the Correlator is instantiated.
   - Correlator Deletion - The function is called when the instance of the Correlator is deleted.

   Refer to Table 3-8 on page 57 for details on Correlator Creation and Deletion.

   Select the option from the `Function Usage` drop down menu.
5. Select the mode in which the function must be called, synchronously or asynchronously. This option is not valid for Composer Built-in functions i.e Built-In functions are always invoked synchronously.

6. The parameters for the function must now be provided. Select the parameters from the Parameter list's pop up menu.
7. Click on `[OK]` to complete the function definition.

To create a Callback function to be called while discarding the event, click on the `Discard` tab and repeat the above steps.

Validating Function Definitions

The details entered in the Function Definition window are validated against the data in XML files. The Built-in functions are validated with the data in `$OV_CONF/ecs/CIB/BuiltInFuncDetails.xml` file. This file should not be edited as the built-in functions provided with ECS are not extensible.

The C and Perl functions are validated with the data in `$OV_CONF/ecs/CIB/UserDevelopedFuncDetails.xml` file. This file is extensible and you can add details about the C and Perl functions that have been developed (for more information, refer to "Configuring UserDevelopedFuncDetails.xml File" on page 103).

For function definitions with fixed number of parameters, Composer validates the number of parameters. For function definitions with variable number of parameters, Composer checks for the presence of minimum number of parameters. For example, the `StoreStr()` function can hold variable number of parameters though the minimum number of parameters is five.

If the number of parameters does not match the expected value, an error message is displayed. You can then make the required modifications to the function definition.

NOTE The data type of the parameter will not be validated.

Step 4: Define New Alarms

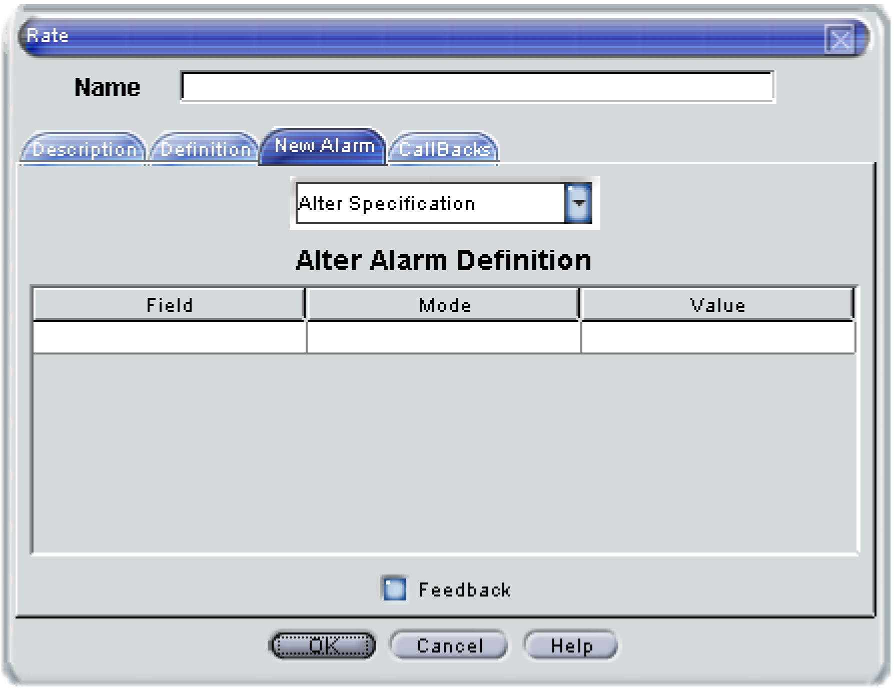
Correlators enable creating and alterating of alarms. New alarms can be created in two ways:

- Alter Specification

  New alarms can be created by altering the existing attributes of alarms. Follow the steps given below to alter the attributes of any alarm.

  1. Click on the `New Alarms` tab in the Correlator window. The New Alarms panel opens. Note that the None option is chosen by default.
  2. Select `Alter Specification` from the drop down menu. The Alter Alarm Definition table is displayed.

Figure 4-7 Alter Alarm Specification



Field	Mode	Value

3. Select the attribute that you would like to alter from the drop down menu in the `Field` column.
4. Select the mode of alteration from the drop down menu in the Mode Column. The two options are
   - replace-replaces the existing event contents
   - append-appends the new values to the existing contents of the attribute
5. Select the variable that is to replace or be added to the attribute contents from the pop up menu.

   Note that any new values to be appended must have already been defined in the Variables section of the Correlator.

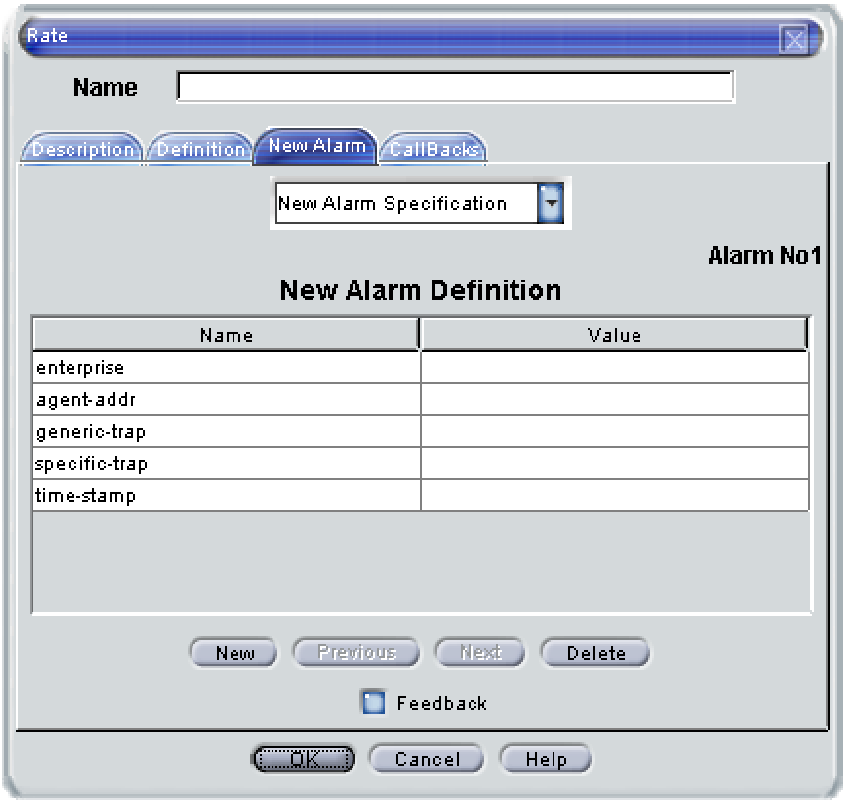
- New Alarm Specification

  New alarms can also be created for the outgoing event. Follow the procedure given below to create a new alarm.

  1. Select `New Alarm Specification` from the drop down menu in the `New Alarms` panel. The `New Alarm Definition` table is displayed.
  2. The `New Alarm Definition` table displays all the mandatory attributes that must be filled. Select values for the fields from the pop-up list displayed.

     The list of attributes are picked up from the configuration file under `<Composer install directory>/CO.conf`. If you need to add more attributes, you must edit the `CO.conf`. Refer to “Define Event Attributes” on page 66 for details on how to edit the configuration file.

Figure 4-8 New Alarm Specification



Name	Value
enterprise	
agent-addr	
generic-trap	
specific-trap	
time-stamp	

3. To add new attributes to the new alarm created, right click the mouse button and select Add. A new row is added to the New `Alarm Definition` table.
4. Notice that the text “`Alarm Number:1`” is displayed on the right hand top corner of the table. The Alarm number indicates the number of the new alarm being created. You can navigate through the list of alarms defined.

   Refer to the table given below to understand the functionality of the other buttons.

Table 4-3 New Alarm Definition

Button Name	Description
`Delete`	Deletes the new alarm.
Feedback	All alarms are fedback into the Composer.
`New`	Creates a New Alarm. Displays a new New Alarm Definition table.
`Next`	Navigates through the alarms created. Displays the contents of the succeeding alarm in the new alarms list.
`Previous`	Navigates through the alarms created. Displays the contents of preceding alarm in the new alarms list.

Step 5: Callback Functions

When a Correlator either discards an alarm or creates a new alarm, the user can optionally choose to invoke a user-defined function, implemented in either ‘C’, PERL or the built-in functions. Parameters to these functions can be chosen from the set of variables defined for the Correlator. Typically the callback functions are used to create audit trails. For example when an event is deleted, a logging function can be invoked. Follow the procedure given below to create a Callback function:

1. Click on the `Callbacks` tab in the Correlator window. The `Callbacks` panel is displayed.
2. Enter the Function Name. This is the name of the Callback function provided by the user.
3. Enter the `Function Description`. Provide a description that will help you identify what the function does.
4. Select the `Function Type` from the drop down menu and the mode in which the function must be called.
5. Select the time at which the Function is to be called from the `Function Usage` drop down menu.
6. To select parameters to the external function, select the attributes from the pop up menu displayed in the Parameters table.
7. To add more parameters to the function, right click the mouse button and select the attributes from the pop up menu.
8. Click on `[OK]` to complete the Callback function definition.

Variables available to be used in Callbacks

All alarms have their attributes available via their corresponding alarm names. Automatic Variables are available for access by the Create and Discard Callback functions. The Create Callback can access the attributes of the new alarm just created via the Automatic Variable `NewAlarm`. The Discard Callback function’s automatic variables are dependent on the Correlation Model chosen. The table below lists the variables.

Table 4-4 Automatic variables available to Discard Callbacks

Model	Automatic variables available to Discard Callback(can be used as parameters to the Callback function)
`Enhance`	None
`Multi-Source`	The attributes of the discarded alarm is available via the automatic variable “Discarded”. (The attributes of the alarms in the set will also be available via their names as usual)
`Rate`	None
`Repeated`	None
`Suppress`	None
`Transient`	The discard Callback is called only for the Fail alarm. The Callback can access all the attributes of the Clear alarm via the automatic variable “Suppressor”
`User-Defined`	None

Step 6: Load Perl and C Library

External functions can be written in Perl or C. The Composer must be supplied with the names of the files where these functions reside. To provide the name of the Perl file to the Composer, follow the given procedure:

1. Select `Correlations->Perl File` from the Correlator Store window. The `Perl File` window is displayed.
2. Enter the name of the Perl file.
3. Click `[OK]` to close the window.

The table below lists the default path for the Perl file containing the external Perl functions. is `$OV_CONTRIB/ecs/external/perl`.

Operating System	**Default path**
HP-UX	`$OV_CONTRIB/ecs/external/perl`
Linux	`$OV_CONTRIB/ecs/external/perl`
Solaris	`$OV_CONTRIB/ecs/external/perl`
Windows	`%OV_CONTRIB%\ecs\external\perl`

To pick up Perl files from a different location, the relative path must be specified in the Perl File window. Refer to "Writing External functions in Perl" on page 105 for information on how multiple Perl files can be referenced.

To set the default C library,

1. Select `Correlations->C Library Name` from the Correlator Store window. The `C Library Name` window is displayed.
2. Enter the name of the C library that contains the external function. The default library for the C function is placed under `$OV_CONTRIB/ecs/external`.

NOTE

You can also specify the library name while defining the function. The function name can be prefixed with the C library name, where this function resides. For example, if a function named `BSCName()`resides in a library called `SNMPlib`, you specify function name as `SNMPlib:BSCName()` in the `Function Name` text box.

Note that multiple libraries can be loaded in this way. The library name entered in the `C Library Name` window is the default library, while the library name specified while defining the function can be a completely different library.

Table 4-5 **C library naming conventions**

Operating Systems	C Library naming convention	Example
HP-UX	`.sl`	`SNMPlib.sl`
Linux	`.so`	`SNMPlib.so`
Solaris	`.so`	`SNMPlib.so`
Windows	`.dll`	`SNMPlib.dll`

3. Click `[OK]` to close the window.

If the Composer encounters an error during Runtime, an error-trap is displayed, describing the problem. Refer to “Troubleshooting the Composer during Runtime” on page 195 for more details.

Managing Correlators

Open an existing Correlator

To open an existing Correlator:

1. Select `File:Open` from the menu. The file browser displays all the Correlator Stores that have been defined.
2. Enter the Correlator Store name in the `File` panel or select the filename from the directory listing. The Correlator Store with all the Correlators is displayed.
3. Select the Correlator which you want to open.
4. Double click on the Correlator. The Correlator window is displayed.
5. Select `[OK]` to close the window.

If you have an existing Correlator Store open, the Composer closes it. If the existing Correlator Store has unsaved changes, the Composer prompts you to save the file.

Modify an existing Correlator

Once you have defined a Correlator, you can modify its properties whenever required. To modify a Correlator:

1. Select the Correlator name from the table in the Correlator Store window.
2. Right click the mouse button and select `Modify` from the menu. This opens the Correlator window.
3. Make the required changes as were done while creating the record.
4. Save the changes.

   Click `[OK]` at the bottom of the window to save the modified changes. This closes the Correlator window and the control is returned to the Correlator Store window.

TIP To modify the correlation record you can also:

1. Select the correlation record that you would like to modify.
2. Double click the mouse button. The Correlation window with all data configured is displayed.

Delete an existing Correlator

To delete a Correlator:

1. Select Correlator to be deleted from the table in the Correlator Store window.
2. Right click the mouse button and select `Delete` from the menu displayed.

   The selected correlation record is deleted from the Correlator Store file.

TIP To delete a Correlator you can also:

1. Select the Correlator you want to delete.
2. Hit the `[Delete]` key.

NOTE Deleting a Correlator sorts the remaining Correlators in alphabetical order according to Correlator Name.

The Composer in the Operator’s mode does not allow creation of new Correlators. Refer to Chapter 6, “Correlation Composer for the Operator,” on page 137 for more information.

Writing External functions in C

This section describes how to write the C function such that it is accessible from within the Composer. The procedure consists of two parts

1. Writing the C function using the guidelines as described in the next section.
2. Create a shared library of the C function(s) and store the shared library in the following location

   For UNIX

   `$OV_CONTRIB/ecs/external`

   For Windows

   `%OV_CONTRIB%\ecs\external`
3. Configuring `fileUserDevelopedFuncDetails.xml` file for ease of use of the function definitions in Composer GUI.

NOTE The shared library is loaded by the runtime, the first time a function within the shared library is invoked (as part of a Correlator). When a shared library is loaded, the function `_Init` will be invoked if it exists. The signature for `_Init` is exactly the same as that of other C functions.

Guidelines for writing a C function

Given below is the skeleton code for a function in 'C'

```
#include <stdio.h>
#include <ECS/GC_Values.h>
int testFunction(int argc,void ** argv,int reqId,int cmdId,  genc_callback *
callback)
{
   int i = 0;
   char * str = NULL;
   char * oid = NULL;
   char myStr[] = "some string ";
   char myOid[] = "1.2.3.4.5.6.7.8.9";
   GC_Values ** retValue = NULL;
   GC_Values * intVal = NULL;
   GC_Values * strVal = NULL;
   GC_Values * oidVal = NULL;
/* Do your own checking here - this example checks if number of arguments is 3 */
   if( argc != 3 )
   {
      /* Improper No. for argumnets */
     /* Allocate the space for returning the err string  back  to Composer */

      retValue = (GC_Values **)calloc(1,sizeof(GC_Values *));

      GC_MAKEVALUE(GC_ERRSTR, "Improper Arguments", strVal);


      if(!strVal)
      {     /* just return */
        callback(reqId, cmdId, 0, NULL);
        return ;
     }

     retValue[0] = strVal;
     /* Do callback to notify the error */
     callback(reqId, cmdId, 1, &retValue);
     return 0;
  }

/* Get the arguments passed to the function. The type of the arguments needs to
be defined by the function writer and its the resposibility of the Composer user
to pass in the correct number and type.*/

/* Do not free these values, will be freed by caller when callback is called */
```

```
    i = *(int *)argv[0];
    str = (char *)argv[1];
    oid = (char *)argv[2];

/* Object ID will be passed as string to the function */

/* Do your processing here */

/* processing is done-time to return back to the Composer*/
/* THIS is the second half */
/* Allocate space for 3 return values - one can return any number of retrun
values - the example return 3 */

    retValue = (GC_Values **)calloc(3,sizeof(GC_Values *));


/* Now create the wrapper to pass back the values to Composer*/

    GC_MAKEVALUE(GC_INTEGER, &i, intVal); /* Integer*/
    if(!intVal)
    {
        /* Do Error handling */
    }

    GC_MAKEVALUE(GC_STRING, myStr, strVal); /* String */
    if(!strVal)
    {
       /* Do Error handling */
    GC_FREEVALUE(intVal);
    }
    GC_MAKEVALUE(GC_OID, myOid, oidVal); /* Object ID */
    if(!oidVal)
    {
       /* Do Error handling */
       GC_FREEVALUE(intVal);
       GC_FREEVALUE(strVal);
    }

/* Set the 3 return values in the wrapper */
    retValue[0] = intVal;
    retValue[1] = strVal;
    retValue[2] = oidVal;

/* Call the callback to give the value back to Composer

1. ReqId
2. cmdId passed as the argument to this function
3. Number of return values i.e. number of elements in the
GC_Values array
4. Address of the GC_Values array */
```

```
        callback(reqId, cmdId, 3, &retValue);
        return 0;
}
```

Basic Structure

The signature of all functions that need to be invoked from the Composer is as given below.

`int func(int argc, void ** argv, int reqId, int cmdId, genc_callback *callback)`

where,

`argc` is the number of arguments passed in

`argv` is an array of pointers to the arguments

`reqId` and `cmdId` are opaque parameters and are used while calling the Callback function.

`callback` is a pointer to a function that needs to be invoked on completion and to pass back any return values to the Composer.

Writing the function

Writing a function consists of three distinct parts

1. the first part is getting the arguments passed from the Composer
2. the second part is the processing
3. the third part is to return a value or a set of values back into the Composer

The first part is getting the arguments passed from the Composer. The function can take any number of arguments of any type. It is the responsibility of the user of the function to configure the Composer correctly to ensure that the correct parameters are passed. In the example above the function expects three parameters, an integer, a string and an Object ID. The pointers to these parameters are in `argv` and are accessed as `argv[0], argv[1], argv[2]`. While accessing these pointers, ensure that they are cast the right type.

Parameters passed MUST NOT be freed by the function. The freeing of the space will be done when the Callback function is invoked.

The table below lists the data type as received by the Composer.

Type passed from the Composer	Type as received by the C function
Integer	Integer
Float	Float
String	Char *
Object ID	Char *
Time	Integer

Once the parameters passed in have been accessed, the function needs to process it. The function indicates completion of processing by invoking the Callback function. This mechanism allows the freedom to the function writer to process either synchronously or asynchronously. In other words the function returning does not indicate function completion. The user can choose to extract the arguments, queue them up for some other thread to process it and return immediately. Post processing, the Callback function can be invoked to return the values to the Composer and simultaneously indicate completion(obviously the call to the Callback will be made from a function other than the one that was originally called). This is useful when(for example) the function needs to go over the network to access data or when databases need to be accessed, both of which take time. If the function being invoked takes very little time, it is suggested that the Callback is called and then returned.

If the function encounters an error at any point during it's processing, the error is indicated by calling the Callback function with the error code as shown in the example.

NOTE The Callback function must be called whether or not the function succeeds or fails.

Returning values back - Returning values consists of four steps

1. allocating space for the return value
2. wrapping the return value

3. marshalling the return values
4. calling the Callback function

Step 1: Allocating space for the return value

Make the following calls to allocate space

```
retValue = (GC_Values **)calloc(X,sizeof(GC_Values *));
```

where X is the number of return values that needs to be returned.

`retValue` is defined like - `GC_Values ** retValue = NULL;`

Step 2: Wrapping the return value

Invoke the provided macro(present in the file `GC_Values.h` under the directory `$OV_HEADER/ecs/ECS` on HP-UX, Solaris and Linux and `%OV_HEADER%\ecs\ECS` on Windows) - as shown below

```
GC_MAKEVALUE(GC_STRING, valToReturn, strVal);
GC_Values * strVal = NULL
```

where,

the first parameter is one of

- GC_INTEGER to return an integer
- GC_STRING to return a string
- GC_OID to return an Object ID(to return an Object ID, the

  `valToReturn` should be a string in the dot notation).

  Example “1.2.3.4”
- GC_FLOAT to return a float

the second parameter `valToReturn` holds the value to be returned

the third parameter `strVal` is a pointer to the macro `GC_Values` (Example, `GC_Values * strVal;`)

Step 3: Marshalling the return values

In Step1, we allocated space. In Step 2, we wrapped the return values. In this step we will tie the two. Assuming there are two values to be returned, do the following

```
retValue[0] = valToReturn1;
```

```
retValue[1] = valToReturn2;
```

where `retValue` is got in Step1 and `valToReturn1` and `valToReturn2` are got is Step2.

Step 4: Calling the Callback function

The last step is to indicate completion of processing and returning the values to be returned. This is done as follows

```
callback(reqId, cmdId, 3, &retValue);
```

where,

`callback` is the pointer to a function which was passed as the fifth parameter and `reqId` and `cmdId` were the third and fourth parameters passed in.

Configuring UserDevelopedFuncDetails.xml File

To eliminate the task of manually entering function details such as name, signature, and call mode while creating a function definition, you can configure the `UserDevelopedFuncDetails.xml` file.

This feature enhances usability as the function details can be accessed readily from the Composer UI. For example, accessing function details in the Function Definition dialog box. The XML file is available in `$OV_CONF/ecs/CIB/` directory. To configure the file:

1. Add function details to the XML file.

   The details to be added for each function are:

Function Detail	**Description**
Name **(FunctionName)**	The function name in the format: *`LibraryName:FunctionName`*
Description **(FunctionDescription)**	Brief description of function behaviour
Signature **(FunctionSignature)**	The function signature. Example: `add` *`int1 int2`* *`int1 int2`* are the integer values to be added.

Function Detail	**Description**
Number of parameters **(No_Of_Args)**	Number of parameters to be added; -1 indicates variable number of parameters
Minimum number of parameters **(Min_No_Of_Args)**	The minimum number of parameters expected. Applicable only if the number of parameters is -1.
Type **(FunctionType)**	Value can be C or Perl.
Usage **(FunctionUsage)**	Value can be one of the following: Default Correlator Creation Correlator Deletion Correlator EventIn
Function call mode **(FunctionCall)**	Value can be Asynchronous or Synchronous

2. Validate the updated XML file.

   The XML file is validated for syntax and semantic errors when Composer starts up. The function definitions are validated with the corresponding XML schema file, `function.xsd`, located in the same directory as the XML file. If no errors are found, the parser loads the definitions into memory.

3. Errors, if any, will be reported on the command line or in a log file (if XPL is enabled).

NOTE

Composer supports this feature only in Developer mode.

If this feature is not used, you can manually enter the function details as before.

Writing External functions in Perl

This section describes how to write a Perl function such that it is accessible from within the Composer. The procedure consists of two parts

1. Writing the Perl function using the guidelines as described in the next section.
2. Store the file in the following location

   For UNIX

   `$OV_CONTRIB/ecs/external/perl`

   For NT

   `%OV_CONTRIB%/ecs/external/perl`

NOTE Only one Perl file can be loaded into the Composer at runtime. If multiple Perl files are required refer to “Support for Multiple Perl files” on page 107.

WARNING **It is strongly recommended that the Perl functions are tested outside the Composer. The embedded Perl interpreter is known to exit on syntax/parse errors.**

Guidelines for writing a Perl Function

The function `orchPerlfunction` in the skeleton given below contains the key elements necessary to write a Perl function. The arguments are passed to the function as an array and individual parameters can be accessed as array elements. Object IDs are passed in as a string in the dot notation format. Example “1.2.3.4”. As shown in the skeleton below multiple return values can be returned. If an Object ID needs to be returned it needs to be encapsulated before returning. Refer to the skeleton below. Include the subroutine `ecdlEncap` given below if you need to return an Object ID.

```
# Function  to encapsulate return values into ECS specific data types
# Do not modify
sub ecdlEncap
{
   my $x=$_[0];
   my $z=$_[1];
   if( "$z" eq "ECS_OID")
  { push @$x,"ECS_OID"; }

}
sub OrchPerlFunction
{
  # Get the arguments to the function
   # my $arg0 = $_[0];
   # my $arg1 = $_[1];
   # my $arg2 = $_[2];

   # Do processing here

  # return value(s) back to the Composer
# Example of returning an int - uncomment the next two lines & modify
     # $retVal = 10
     # $retVal
# Example of returning a string - uncomment the next two lines & modify
     # $retVal = "Hello World";
     # $retVal;
# Example of returning a multiple values - uncomment the next four lines & modify
    # @retVal ;
    # @retVal ;
    # $retVal[0] = "Hello World";
    # $retVal[1] = 10;
    # $retVal[2] = "This is the 3rd return value";
    # @retVal;
# Example of returning a Object ID type - OID . Uncomment the next 3 lines &
# modify
    # @retVal = ("1.2.3.4");
    # ecdlEncap(\@retVal, "ECS_OID");
    # return (\@retVal);
# Example of multiple values, including OID. Uncomment the next 5 lines and
# modify
   # @retVal0 = ("1.2.3.4");
   # $retVal1 = 1;
   # return an Integer
   # $retVal2 = "Hi there";
   # return an string
   # ecdlEncap(\@retVal0, "ECS_OID");
```

```
     # return(\@retVal0, $retVal1, $retVal2);
     # return a Object ID, int and a string
}
```

Support for Multiple Perl files

Composer supports multiple Perl files but it doesn't support multiple perl files with more than one MAIN routine (BEGIN block). Consider two files `p.pm` and `t.pl` as below,

p.pm

```
sub f {
        my $rv = "HELLO THERE!!";
        return $rv;
}
1;
```

t.pl

```
BEGIN{
push(@INC, 'd:\/openview\/contrib\/ecs\/external\/');

}
use p;
```

Notice that,

- the Perl file `t.pl` `includes` the file `p.pm` using the keyword `use`
- only t.pl has the BEGIN block. The Composer expects Perl files to be written in this manner

There should be one main perl file with a BEGIN block, which does nothing or probably can do some initialization tasks and all the other Perl files need to be 'included' in the main Perl file using keyword the `use.` (as `p.pm` is included in the file `t.pl` using the statement `use p;`).

Consider two users creating individual Perl files. The first user develops a set of Perl functions in `user1.pm` and the second user develops a set of Perl function in `user2.pm`. Both these files must be included in the file `main.ovpl` using the statements `use user1;` and `use user2;`.

Now, only the file `main.ovpl` needs to be referred from the Composer. Please note that if the files user1.pm and user2.pm are in a different location than the file `main.ovpl,` then the file `main.ovpl` must specify the statement

`push(@INC, <`*`location of perl file(s)`*`);`

This statement enables the inclusion of search path(s) of Perl files.

If `user1.pm` and `user2.pm` have clashing function names then the scope resolution operator (`::`) needs to be used while making the function call like `user1::f()`.

IMPORTANT All Perl files other than the main Perl file must have the extension `.pm`.

Configuring UserDevelopedFuncDetails.xml File

To eliminate the task of manually entering function name and other details whenever you create a function definition in Composer, you can configure the `UserDevelopedFuncDetails.xml` file. For more information, refer to "Configuring UserDevelopedFuncDetails.xml File" on page 103.

User Defined Correlation

The User-Defined Correlator Template can be used to implement a requirement when none of the other Correlator Templates, either by itself or in a combination can meet the requirement. The procedure to define the User-Defined Correlation is similar to defining other Correlator Templates, in the sense that there is an Alarm Definition section, the New Alarm section and the Callback section.

Concept of User-Defined function - The model for this correlation is as follows:

The Input function is called when the event satisfies the Alarm Signature and Advanced Filters. The Input function can be written in Perl, C or the built-in functions like makelist. However, it is mandatory that the return value be in the following format

`flag``, `*`window`*`, `*`alarm mask`*`, optional values`

where,

`flag` is mandatory and indicates the action to be taken on the alarm. Multiple values can be provided by bitoring the flags.

`window` is mandatory and is valid only if the flags have one of the HOLD, WEAKHOLD, PSEUDOHOLD set. It indicates the time in seconds after which the output function will be invoked.

`alarm mask` is mandatory and is used to control the set of alarms to be created. In the New Alarm section several alarms can be defined. However, in many cases only a subset to these new alarms need be created. This mask is used to control the set of alarms to be created. For example, if in the New Alarm section, five alarms have been defined and the alarm mask is set to 3(last two bits are set) then the first and second alarms are created.

NOTE This is valid only if the create option is chosen as part of the flag, otherwise this value is ignored.

To create all alarms defined, use a mask of -1.

`optional values` - Any other values (other than the mandatory values)returned will be bound to a variable called `InputRetVal`, that is, if the function returned HOLD, 5, 10 then `InputRetVal` is bound to the value 10 and can be used like any other variable. If the function returned more than 1 optional value, like HOLD, 5, 10, 20, 30 then InputRetVal is bound to a list that will contain 10, 20, 30. Individual elements can be accessed using the built-in function `getByIndex`.

`flag` can take one or more of the following values.

`ALTER`	Alter the event as specified in the Alter Alarm specification
`CREATE`	Create a new event as specified in the New Alarm specification
`DISCARD`	Discard the event
`HOLD`	Hold the event for a time period as specified in the window parameter, after which time the Output function specified is invoked. If `Hold` is specified then the event is held for window period regardless of other Correlators outputting the same event. At the window period, the output function is invoked and it's return value will determine the action to be taken on the event.
`PASSTHROUGH`	Output the event. Note that the event will be output ONLY if no other Correlator decides to DISCARD it or HOLD.
`WEAKHOLD`	Hold the event for the time period as specified in the *`Window`* parameter. WEAKHOLD indicates that if other Correlators output the event, the event will be output, nevertheless a copy of the event is held and the output function is invoked after *`Window`* seconds. Contrast this with HOLD where the event is NOT output for atleast window seconds. Weakhold is typically used to invoke the output function at the end of the window period, to send a new alarm(either by creating or altering).
`PSEUDOHOLD`	If specified, the event is NOT held, however the output function is called after window seconds. If PSEUDOHOLD is specified, then the output function cannot return ALTER or CREATE. If you need to

create/alter alarms in the output function then you must use either WEAKHOLD or HOLD. Use PSEUDOHOLD when the output function needs to be called after window period typically to do cleanup.

NOTE The HOLD, PSEUDOHOLD and WEAKHOLD flags can be used only by the Input function.

The table below provides the value of the flag.

Table 4-6 **Flag values**

flag name	**Value**
HOLD	1
DISCARD	2
ALTER	4
PASSTHROUGH	8
CREATE	16
PSEUDOHOLD	32
WEAKHOLD	64

Output functions are invoked when the flag returned by the Input function is either HOLD, PSEUDOHOLD or WEAKHOLD. The function is called after *window* seconds specified in the Input function. The return value of the Output function must be of the form

flag, alarm mask, optional values

where,

flag is mandatory and indicates the action to be taken on the alarm

alarm mask is mandatory and is used to control the set of alarms to be created. Refer Input function for details.

optional values are any other values returned by the function and is bound to a variable called OutPutRetVal. The OutPutRetVal is accessed in a similar manner to the variable InputRetVal discussed earlier.

Writing External Functions to be called as the Input/Output functions of a User-Defined correlation

The process of writing a function in C to be called as the Input or Output function in the User-Defined Correlation, is exactly the same as that of writing an external C or Perl function. The only difference is in the return values. As explained above the Input function must return atleast two values - the `flag` and the `window period`, while the output function must return atleast one value - which is the `flag`. The skeleton for the Input and Output functions are given below. Note that the values for the flags are defined in `GC_Values.h(present` under the directory `$OV_HEADER/ecs/ECS` on HP-UX, Solaris and Linux and `%OV_HEADER%\ecs\ECS` on Windows)

```
#include <stdio.h>
#include <ECS/GC_Values.h>
int InputFunction(int argc,void ** argv,int reqId,int cmdId,genc_callback *
callback)

{
int flags = GC_WEAKHOLD|GC_PASSTHRU;
int window = 300 /* window period of 300 seconds */
int  arg1 = 0, createMask =0;
char * arg2 = NULL;
GC_Values ** retValue = NULL;
GC_Values * flagVal = NULL;
GC_Values * maskVal = NULL;
GC_Values * windowVal = NULL;

/* Do your own checking here */

/* Get the arguments passed to the function - assuming 2 parameters are passed in
*/
arg1 = *(int *)argv[0];
arg2 = (char *)argv[1];

/* Now that you have the parameters passed in, do your processing here */

/* processing is done - time to return back to the Composer */

/* THIS is the second half */

/* Allocate space for 3 return values - one can return more than 3 */
       retValue = (GC_Values **)calloc(3,sizeof(GC_Values *));

/* Now create the wrapper to pass back the values to Composer*/
```

```
        GC_MAKEVALUE(GC_INTEGER, &flag, flagVal); /* Integer */
        GC_MAKEVALUE(GC_INTEGER, &createMask, maskVal);
       GC_MAKEVALUE(GC_INTEGER, &window, windowVal);

        /* Set the 3 return values in the wrapper  */
       retValue[0] = flagVal;
       retValue[1] = windowVal;
      retValue[2] = maskVal;

           callback(reqId, cmdId, 3, &retValue);
           return 0;
}
/* Skeleton for the Output function */
int OuputFunction(int argc,void ** argv,int reqId,int cmdId,genc_callback *
callback)
{
    int flags = GC_CREATE;
    int createMask=-1; /* create all new alarms defined */
    int  arg1 = 0;
    GC_Values ** retValue = NULL;
    GC_Values * flagVal = NULL;
    GC_Values * maskVal = NULL;


/* Do your own checking here */

/* Get the arguments passed to the function- assuming 1 argument is passed in  */

 arg1 = *(int *)argv[0];

/* Now that you have the parameters passed in, do your processing here */

/* processing is done - time to return back to the Composer */

/* THIS is the second half */

/* Allocate space for 2 return values - one can return more than 2 */
    retValue = (GC_Values **)calloc(2, sizeof(GC_Values *));

/* Now create the wrapper to pass back the values to Composer*/

   GC_MAKEVALUE(GC_INTEGER, &flag, flagVal); /* Integer */
   GC_MAKEVALUE(GC_INTEGER, &createMask, maskVal); /* Integer */

/* Set the 2 return values in the wrapper */

retValue[0] = flagVal;
retValue[1] = maskVal;
callback(reqId, cmdId, 1, &retValue);
return 0;

}
```

Merging Correlator Store files

The `csmerge` tool is used to merge two Correlator Stores. Correlator Stores may need to be merged in several cases. For example, when a Correlator in the production environment needs to be updated with the latest revision of the Correlator or when newly developed Correlators need to be added into the production environment.

Within a Correlator Store, no two Global Constants or no two Correlators can have the same name. Two Global Constants or Correlators with the same name but with different values or Correlation logic is called a clash. The merge will be automatic if there is no clash in names. However, if there is a clash, then external input is required to continue with the merge. External input is provided either interactively or by specifying them in the Configuration file.

The tool is available under

- $OV_CONTRIB/ecs for HP-UX, Solaris and Linux
- %OV_CONTRIB%\ecs for Windows

NOTE The tool is implemented as a Perl script and requires a minimum revision of Perl 5.6.

The `csmerge` tool recognizes the following options:

csmerge -namespace NameSpace.conf *<final Correlator Store name>*

csmerge -rm_desc *<Correlator Store name> <final Correlator Store name>*

csmerge *<file1> <file2> <final Correlator Store name>* -config *<configuration filename>*

The `csmerge` -h command summarizes the usage of `csmerge`. You can give only one command at a time. The `csmerge` command ignores all commands except the first.

Merge Correlator Stores that are specified in the Namespace

Correlators Stores listed in the NameSpace can merged by specifying the name of the NameSpace file. All Correlators from the Correlator Stores are prefixed with the Logical name (as mentioned in the NameSpace file) of the Correlator Store as <*Logical Name*>_< *Correlator Name*> in the final Correlator Store. In the event there is an overlap of names of Global Constants, the Global Constants are also prefixed with the Logical Name of the Correlator Store. Hence it is important that Logical Names for Correlators be unique.

When `csmerge` is invoked with the `-namespace` option, all Correlator Stores are locked to enable merging. If the locking fails even for one of the Correlator Stores, then the merge process fails.

To merge the Correlator Stores that are listed in the Namespace file, type,

csmerge -namespace *<Namespace filename> <final Correlator Store name>*

where,

Namespace filename is the name of the Namespace file from which the Correlator Store files will be picked

final Correlator Store name is the name of the merged Correlator Store

Remove User Description from Correlator Store

To remove the user description from a Correlator Store file, type

csmerge -rm_desc *<Correlator Store name> <destination Correlator Store name>*

where,

Correlator Store name is the name of the Correlator Store from which the user description is to be removed

destination Correlator Store name is the name of the Correlator Store without the user description

Merge Correlator Stores

Correlator Stores created and not listed in the NameSpace Configuration file can also be merged to a single Correlator Store. To merge two these Correlator Stores, type,

`perl csmerge <file1> <file2> <mergedfile> -config <configuration filename>`

where,

`file1` and *`file2`* are the Correlator Stores to be merged

`mergedfile` is the resultant file after merger

`configuration filename` is the file, which if present, specifies which values will be considered while merging the Correlator Stores. When this option is specified, the user in NOT prompted for input and all specifications is picked from the Configuration file.

For a given clash, one of following can happen:

- the definition is picked from File1
- the definition is picked from File2
- both definitions are picked, but the name for one of them needs to change

In the interactive mode(where is there is no Configuration file) the user is prompted for input to decide which of the above needs to happen. In the non-interactive mode, the Configuration file is used to resolve any clashes.

Description of Configuration File

The format of the Configuration file is
`decision-tag:<list of comma separated names>`

where,

`decision-tag` is one of the following:

- `Global_Constant_from_File1` - The Global Constant from File1 is used into the `mergedfile`
- `Global_Constant_from_File2` - The Global Constant from File2 is used into the `mergedfile`
- `Correlator_from_File1` - The Correlator from File1 is used in the `mergedfile`

- `Correlator_from_File2` - The Correlator from File2 is used in the `mergedfile`

It is mandatory that punctuation rules be followed while creating the Configuration file:

- Names must be separated by commas
- Every decision-tag must begin on a new line
- The decision-tag and the list of names must appear on the same line
- The decision-tag and the list of names must be separated by a ':'
- Comments must be on new lines, prefixed with a # sign. So, each line starting with # in the first column is read as a comment.

Example

Assume File1 has a Global Constant called "Clash" with a value of 100 while File2 also has a global Constant "Clash" with a value of 200. The two possibilities of why this happens is

The value of Global variable was deliberately changed to 200, in which case the only global that will be in the merged file is "Clash" with a value of 200. In this case the Configuration will look like

```
Global_Constants_from_File2:Clash
```

The other possibility is that the names being the same was mere coincidence, in which case both need to be added to the merged file and name of one of the variables needs to change. In this case the Configuration file is empty but the file needs to be present.The merge tool automatically changes one of the names and adds both to the merged file.

Example

Shown below is an example of a Configuration file

```
Global_Constant_from_File1:a,b

Global_Constant_from_File2:x,y

Correlator_from_File1:i,j

Correlator_from_File2:m,n
```

The above specifies to the merge tool that:

- if there is a clash for the Global Constants `a` or `b`, then the Global Constants a and b in the merged file are taken from File1

- if there is a clash for the Global Constants `x` or `y`, then the Global Constants `x` and `y` in the merged file are taken from File2
- if there is a clash for the Correlators with names `i` or `j`, then the Correlators `i` and `j` in the merged file are taken from File1
- if there is a clash for the Correlators with names `m` or `n`, then the Correlators `m` and `n` in the merged file are taken from File2

Additionally it also specifies that if there is a name clash other than those specified in the Configuration file, then both will be used in the merged file after renaming one of them. The tool generates a unique name by appending a '_' to the name from File2. For example, if there is a name clash for a Correlator with name `z`, then both Correlators are added to the merged file. The Correlator from File2 is renamed to `z_`.

NOTE The default C library and the Perl filename is always taken from File2.

5 Correlation Composer for the Developer

This chapter explains the Composer's Developer mode the related concepts and terminology used. It also provides an overview of the administrative tasks that need to be executed to ensure that the Composer runs effectively and efficiently.

NOTE All tasks described in this chapter should be executed by the Correlator Store Developer unless otherwise mentioned.

Composer in the Developer mode

The Developer creates/modifies correlation logic for the network environment and sets up access rights to Operators. This section describes the tasks that must be performed by the Developer. The Developer's administrative tasks can be categorized as follows:

- Create or modify Correlator Store(s)

  The primary responsibility of the Developer is to create/modify the Correlator Store file. To create a Correlator Store file, refer to Chapter 4, "Developing Correlators with Composer," on page 69.

- Operator Access Configuration

  The Developer defines Operator access by maintaining NameSpace and Security files.

- Create the Deploy Configuration file

  The Developer creates the Deploy Configuration file required to deploy the correlation logic into the ECS engine.

Starting the Composer in Developer mode

To start the Composer in the Developer's mode, type

```
ovcomposer -m d
```

See Also *ovcomposer* manpage

Planning Operator's Profiles

This section provides easy to follow steps to plan the operator profile configuration for the Composer. Use it in conjunction with the section "Providing User Access", to set up Operator profiles. You can plan your Operator profile based on the examples in this chapter.

Planning the configuration can be divided into the following simple steps:

Step 1:	Creating Correlator Stores.
Step 2:	Listing Correlator Stores.
Step 3:	Creating NameSpace and Security files.
Step 4:	Creating the Deploy Configuration file.

Step 1: Creating Correlator Stores

The Developer creates Correlator Store files in such a way that correlators defined for set environments are grouped logically, that is, all Correlators logically bound are put into one Correlator Store.

Step 2: Listing Correlator Store

Correlator Store files defining correlation logic must be made accessible to Operators. A list of Correlator Stores to be displayed to the Operator is created.

Step 3: Creating NameSpace and Security files

NameSpace

A NameSpace file contains a list of Correlator Store files, grouped logically together to define Operator profiles. This grouping is specifically used to assign access permissions to the Correlator Store files to Operator profiles and has no other relevance. The list of Correlator Stores specified in this file decides the area of operation within which the Operator can work.

The NameSpace file is a simple ASCII file that can be edited using any standard text editor. This file is a listing of name-values pairs of the Correlator Store name versus the relative path of the location of this Correlator Store.

NOTE To create/edit the NameSpace file, the user must have `root` access on the machine where Composer is installed.

The general syntax for a NameSpace file is

```
<Logical Name1>=<Location of Correlator Store file>
<Logical Name2>=<Location of Correlator Store file>
.
.
.
```

where,

<Logical Name1>, <Logical Name2>	are the logical names of Correlator Store files as will be displayed in the Composer when started in the Operator mode.
<Location of the Correlator Store file>	is the location of the Correlator Store file relative to the directory `$OV_CONF/ecs/CIB`.

Following is a sample of the NameSpace file:

```
#comment line:path relative to the $OV_CONF/ecs/CIB directory
ATM=ATM/atm.fs
OV=OV/ov.fs
CISCO=CISCO/cisco.fs
```

Rules while creating a NameSpace file

The NameSpace file must be edited following the rules provided, in order that access to the Correlator Store files is successful.

1. The location of the Correlator Store file on the right hand side of “=” is *always* relative to the directory `$OV_CONF/ecs/CIB`.

2. Correlator Store files present above the directory `$OV_CONF/ecs/CIB` are not accessible. They must be present under a subdirectory (typically, with the same name as that of the Correlator Store) or under the directory `$OV_CONF/ecs/CIB`.
3. No blank space allowed before and after the "=" sign.
4. Every entry for a logical name is made on a separate line.
5. Logical names of Correlator Stores *must* be unique.
6. All file location paths specified must be on a single line and *must* not flow over to the next line.
7. Ensure that the file is saved with the extension `.conf` *always*.
8. All comments are preceded by the hash (#) symbol.

IMPORTANT Ensure that the NameSpace file referenced in the Deploy Configuration file (refer to "Step 4: Creating the Deploy Configuration file" on page 129) is the same file passed with the `-N` option when Composer is started (refer *ovcomposer* manpage). If the filenames differ, then it leads to creating one set of Correlator Stores and deploying a completely different set of Correlator Stores.

Security File

The Security File of the Composer contains a list of fields/parameters that can be edited by the Operator. Every Correlator Store file created, has a corresponding Security file associated and is stored in the same directory as that of the Correlator Store file.

A default Security file is created for every Correlator Store file saved the first time. This file is stored in the same directory as the Correlator Store file. The default Security file(<*Correlator Store filename*>`.sec`, where <*Correlator Store filename*> is the name of the Correlator Store for which the security file is created), allows all values of Parameters in the Alarm Definition section of all Correlators to be edited. The Security file contains a list of editable fields of Correlators. Only the values of these fields can be changed by the Operator. The Security file is a simple ASCII file and can be edited using any standard text editor.

The general syntax of the Security file is as follows:

```
ALL_TEMPLATE=TOK_LIST
ALL_TEMPLATE=CORRELATOR_STATUS
GLOBAL_CONSTANT=GC_LIST
CORRELATOR_TEMPLATE=TOK_LIST
CORRELATOR_NAME=TOK_LIST
```

where,

`ALL_TEMPLATE=`*`TOK_LIST`*	All Correlator Templates have access to edit values of parameters listed. *`TOK_LIST`* can be any token identifier listed in Table 5-2 on page 128. However, any other condition specified in the Security file will *not* be overridden by this statement.
`ALL_TEMPLATE=CORRELATOR_STATUS`	With this condition the Operator can choose to enable or disable the Correlator to participate in correlation. If this condition is not specified, the Operator is not allowed to enable/disable Correlators. However, by default all Correlators will participate in correlation if it is already enabled.
`GLOBAL_CONSTANT=`*`GC_LIST`*	List of Global Constants whose values can be edited. `GC_LIST` is the list of Global Constants whose value can be edited.
`CORRELATOR_TEMPLATE=TOK_LIST`	List of parameters for the specific Correlator Template type for which values can be edited. *`TOK_LIST`* can be any token identifier listed in Table 5-2 on page 128 and *`CORRELATOR_TEMPLATE`* is the Correlator template names as listed in Table 5-1 on page 127.
`CORRELATOR_NAME=TOK_LIST`	List of parameters for the specific Correlator whose values can be edited. *`CORRELATOR_NAME`* is the name of the

Correlator and *TOK_LIST* is any token identifier as specified in Table 5-2 on page 128.

Following is a sample of the Security file:

```
OV_Chassis_Cisco=NEW_ALARM
USER_DEFINED=ALARM_SIGNATURE
ALL_TEMPLATE=WINDOW
```

From the above example, the following can be interpreted:

- For the Correlator OV_Chassis_Cisco, parameters defined for New Alarm creation can be edited.
- For all User Defined correlators the values in the Alarm Signature section can be edited.
- For all Correlator Templates other than User Defined and the Correlator OV_Chassis_Cisco, the value for the Window parameter can be edited.

IMPORTANT

To edit the value of Window parameter for the USER_DEFINED template or the OV_Chassis_cisco Correlator, it must be specified explicitly. Type,

```
OV_Chassis_cisco=NEW_ALARM,WINDOW
USER_DEFINED=ALARM_SIGNATURE,WINDOW
```

Rules while creating the Security File

The Security file must be created following the rules given below:

1. Every condition to be specified must be made on a separate line.
2. Token parameters are separated by commas.
3. No blank space is allowed before and after the commas used as seperators for token identifiers.
4. All comments are preceded by the hash (#) symbol.
5. Save the file as <*Correlator Store filename*>.sec *always* in the same directory where the Correlator Store is stored.

6. The order of precendence for conditions in the Security file is the Correlator Name, Correlator Template Type and finally the condition for all templates.

   This precedence is arrived at wholly to provide complete security and to have a rigid control on the Correlator Store.

7. **Editing values of Global Constants**

   To edit the values of Global Constants, use the token identifier GLOBAL_CONSTANT. For example, if you want to provide permission to the user to edit the values of the Global Constants `pi`, `timeout` and `createtime`, type,

   ```
   GLOBAL_CONSTANT=pi,timeout,createtime
   ```

8. **Editing values specific to Correlator Templates**

   Provide appropriate token identifiers to make specific changes to values of attributes/variables in Correlators. All identifiers must be specified in upper case. Follow the conventions provided in the tables below while creating the Security file.

Table 5-1 Token Identifiers

Parameter	Token Identifier
All Correlator Templates	ALL_TEMPLATE
Enhance Correlator Template	ENHANCE
Global Constants	GLOBAL_CONSTANT
Multi Source Correlator Template	MULTI_SOURCE
Rate Correlator Template	RATE
Repeated Correlator Template	REPEATED
Suppress Correlator Template	SUPPRESS
Transient Correlator Template	TRANSIENT
User Defined Correlator Template	USER_DEFINED

Table 5-2 Token Identifiers for TOK_LIST

Parameter	Token Identifier
Advanced Filter	ADVANCED_FILTER
Alarm Signature	ALARM_SIGNATURE
All parameters	ALL_PARAM
Alter Alarm parameters	ALTER_ALARM
'Clear Alarm' of Transient Correlator Template	CLEAR_ALM
Count of number of alarms of Rate Correlator Template	COUNT
Create Callback function parameters	CRT_CALLBACK
Correlator Description	DESCRIPTION
Discard Callback function section	DIS_CALLBACK
Discard alarm in Rate Correlator Template	DISCARD
Discard Duplicate in Repeated Correlator Template	DISCARD_DUP
Discard Immediately in Repeated Correlator Template	DISCARD_IMD
Discard alarms on set completion in Multisource Correlator Template	DISCARD_ON_SET
Enable Threshold of Transient Correlator Template	ENABLE_THR
Enhance the alarm always of Enhance Correlator Template	ENHANCE_ALWAYS
'Input Function' in User Defined Correlation	INPUT_FUN

Table 5-2 Token Identifiers for TOK_LIST

Parameter	Token Identifier
Message Key	MESSAGE_KEY
New Alarm parameters	NEW_ALARM
'Output Function' in User Defined Correlation	OUTPUT_FUN
Participate In Other Correlation of Suppress Correlator Template	PAR_OTHCORR
Wait for Set completion in Multi Source Correlator Template	SET
Threshold Count of the Transient Correlator Template	THR_CNT
'Threshold Window' of Transient Correlator Template	THR_WIN
Variables	VARIABLES
Want Original alarm of Enhance Correlator Template	WANT_ORIGINAL
Time period	WINDOW

Step 4: Creating the Deploy Configuration file

The Operator is finally responsible to ensure that the correlation logic be loaded into the ECS engine. Refer to Chapter 6, "Correlation Composer for the Operator," on page 137 for a detailed procedure. Composer provides this facility by the Deploy feature. However, the Deploy configuration file is maintained by the Developer.

The deploy procedure invokes the `csdeploy` and `csmerge` scripts. These scripts merge the Correlator Store files, removes user description from the merged Correlator Store and then loads the file into the ECS engine. These scripts can also be separately executed from the command prompt. For more information on how Correlator Stores are merged refer to "Merging Correlator Store files" on page 114.

The deploy procedure refers to the Deploy Configuration file, that constitutes the following:

- Name of the Correlator Store file after merge
- Path to where the NameSpace file for the associated Correlator Store file(s) is present.
- Name of the logfile to which the logs while merging are written into.
- ECS Engine Instance to which the merged Correlator Store is loaded.
- Logical name of the merged Correlator Store file.

Creating and Updating Deploy Configuration files

The Deploy Configuration file contains information required by the ECS engine at the time when the Correlator Store files are loaded into the ECS engine. The Deploy Configuration file is an ASCII file to which the above information can be added. A sample of the Deploy Configuration file is shown below:

```
#Following is the default configuration file for the deploy
operation from composer GUI in standalone operator Mode and NNM
CMG Mode.

#SUPPORT_DEPLOY_ON_GUI - determines if the deploy should be
supported from the GUI.(Not implemented at the time of this
release)

#FINAL_CS_NAME - path name of the merged Correlator Store to
which all the correlator store files configured in
NameSpace.conf file are merged in to.

#NAMESPACE_FILE - path name of the NameSpace.conf configuration
file used for deploy operation.

#MERGE_LOG_FILE - path name of the log file where the merge
process logs are kept.

#CS_LOGICAL_NAME - logical name of the correlator store loaded
in the engine.

#ENGING_INSTANCE - instance number of the ECS Engine to which
the correlator store should be loaded.

SUPPORT_DEPLOY_ON_GUI=yes
FINAL_CS_NAME="$OV_CONF/ecs/circuits/Composer.fs"
NAMESPACE_FILE="$OV_CONF/ecs/CIB/NameSpace.conf"
```

```
MERGE_LOG_FILE="$OV_LOG/ecs/csmerge.log"
ENGINE_INSTANCE=1
CS_LOGICAL_NAME=Composer
```

Rules while editing the Deploy Configuration file

- A # sign precedes each comment. All text from the start of the comment to the end of the current line is ignored.
- File locations are always specified with the absolute path. Environment variables can also be used while specifying file locations.
- File locations must be enclosed within quotes.
- No blank space allowed before and after the "=" sign.
- Parameters that must be given values are

`SUPPORT_DEPLOY_ON_GUI`	The user is given the option to choose enabling of deploy of the merged Correlator Store via the GUI. This feature is not supported at the time of this release.
`FINAL_CS_NAME`	Name of the merged Correlator Store file
`NAMESPACE_FILE`	Name of NameSpace file from which the Correlator Stores are picked up from.
`MERGE_LOG_FILE`	Name of the logfile to which the logs of the Correlator Store merge are written into.
`ENGINE_INSTANCE`	The ECS Engine instance number for which the Correlator Store file will be loaded.
`CS_LOGICAL_NAME`	Logical name of the merged Correlator Store.

IMPORTANT Ensure that the NameSpace file referenced in the Deploy Configuration file (refer to "Step 4: Creating the Deploy Configuration file" on page 129) is the same file passed with the `-N` option when Composer is started (refer *ovcomposer* manpage). If the filenames differ, then it leads to creating one set of Correlator Stores and deploying a completely different set of Correlator Stores.

Configuring the Operator

Each Operator must have the following information set up for access to the Composer:

- NameSpace file
- Security file associated to the Correlator Store
- Deploy Configuration file

In the NNM Environment

For details on editing NameSpace and Security files in the NNM environment refer to Chapter 10, "Correlation Composer for NNM," on page 205

Follow the steps given below to provide access to a Operator to access the Correlator Store files.

Step 1

Create the NameSpace file

A default NameSpace file is available at `$OV_CONF/ecs/CIB`. To override the specifications present in the NameSpace file, do the following:

1. Copy the default NameSpace file to any local directory.
2. In the NameSpace file, list the names of Correlator Stores and the path of the Correlation Store (relative to $OV_CONF/ecs/CIB).

   Refer to "Rules while creating a NameSpace file" on page 123.
3. Save the file with extension `.conf`.

Step 2

Create the Security file

A default Security file is created when the Correlator Store file is saved the first time. This file is present in the same directory as the Correlator Store. To override the specifications present in the Security file:

1. List the token identifiers and the parameters that can be edited. Refer to "Rules while creating the Security File" on page 126.
2. Save the file as <*`Correlator Store filename`*>`.sec`. Ensure that the file is stored in the same directory where the Correlator Store file is stored.

IMPORTANT

It is the responsibility of the Developer to ensure that correct permissions are provided to the file, so that this file is not overwritten or edited erroneously.

Step 3: **Create the Deploy Configuration file**

A default Deploy Configuration file is available at `$OV_CONF/ecs/CIB`. To override the specifications present in this file:

1. Copy the default Deploy Configuration file to any local directory.
2. Edit the file with values /names specific to your environment.
3. Save the file with the extension `.conf`.

The newly created NameSpace and Deploy Configuration files are bound together based on the entry `NAMESPACE_FILE` in the Deploy Configuration file. Hence, ensure that the correct NameSpace filename is provided in the Deploy Configuration file.

IMPORTANT

After creating the configuration files for your environment, ensure that the correct filenames with locations are specified at the time of startup of Composer (refer to *ovcomposer* manpage). This is mandatory because if no files are specified, then the Composer picks up the default configuration files.

Deploying the Correlator Store

The Developer deploys the Correlator Store into the ECS engine. Before deploying, make a copy of the existing `NameSpace.conf` file, rename it, and update it to contain the list of Correlator Stores you want to deploy.

The Deploy Configuration file (default file is `$OV_CONF/ecs/CIB/Devdeploy.conf`) should then be updated to refer to the newly created namespace file.

NOTE The above changes are required only if you do not want to disturb the existing configuration details in the `NameSpace.conf` file.

To load the Correlator Store file into the ECS engine, do one of the following:

- Ensure that all Correlator Stores have been saved and closed. Select `Options->Deploy`.
- Click the `Deploy` icon.

The Deploy Status window appears and displays one of the following:

- If the deploy is successful, the Deploy Status window indicates success.
- If an error occurred, the `Deploy Status` window indicates failure. To view the details of the error, select `[Details]` in the `Deploy Status` window. To close the window, select `[OK]`.

Deploy from command prompt

The Correlator Stores can be deployed from the command prompt also using the `csdeploy.ovpl` script provided. The `csdeploy.ovpl` script refers to the Deploy configuration file required by the Composer. Refer to “Step 4: Creating the Deploy Configuration file” on page 129 for details on the Deploy configuration file. The `csdeploy` script resides in the `$OV_BIN` directory. To deploy the Correlator Store, type

`csdeploy.ovpl -p <Deploy Configuration filename>`

where,

<*`Deploy Configuration filename`*> is the name of the Deploy Configuration file. If no filename is specified, the default Deploy Configuration file `$OV_CONF/ecs/CIB/Devdeploy.conf` is selected.

The `csdeploy.ovpl -h` command summarizes the usage of `csdeploy`.

6 Correlation Composer for the Operator

This chapter describes:

- The role of the Operator and access rights.
- The procedure to access the Composer for the Operator.

Composer in the Operator mode

Correlation logic is developed by the Correlator Store Developer who is also responsible to provide access rights to the Operator. The Operator has limited access on the Correlator Store files. These rights are governed by the information provided in the Security and Namespace files. The Operator:

- Cannot create new Correlators and hence no new Correlator Stores too.
- Has access only to those files as specified in the NameSpace file. This file is created and maintained by the Developer. Refer to “Planning Operator’s Profiles” on page 122 for more details. Only those files specified in the NameSpace files will be visible to the Operator in the NameSpace table in the Composer.
- Can edit values of only those parameters that are specified in the Security file. This file is created and maintained by the Developer. Refer to “Planning Operator’s Profiles” on page 122 for more details.
- Can enable/disable Correlators in Correlator Stores.

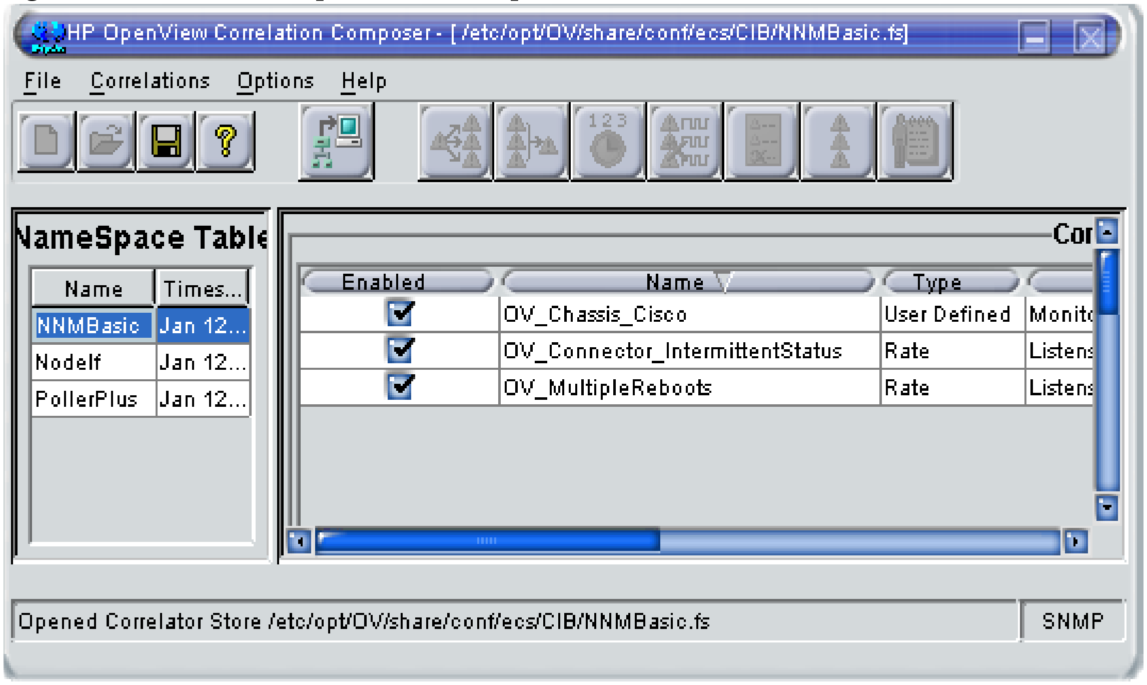
Starting the Composer in Operator mode

To start the Composer in the Operator’s mode, type

```
ovcomposer -m o
```

The Composer in the Operator’s mode opens with the list of Correlator Store files and the last modified time of the Correlator Store in the NameSpace table.

Figure 6-1 Composer in the Operator's mode


HP OpenView Correlation Composer - [/etc/opt/OV/share/conf/ecs/CIB/NNMBasic.fs]
File
Correlations
Options
Help
NameSpace Table
Cor

Name	Times...
NNMBasic	Jan 12...
NodeIf	Jan 12...
PollerPlus	Jan 12...

Enabled	Name	Type	
✓	OV_Chassis_Cisco	User Defined	Monit
✓	OV_Connector_IntermittentStatus	Rate	Listens
✓	OV_MultipleReboots	Rate	Listens

Opened Correlator Store /etc/opt/OV/share/conf/ecs/CIB/NNMBasic.fs SNMP

This section describes the functionality provided by the Composer in order to maintain security while editing Correlator Store files. The main functionality offered to Operators are:

- Mutual Exclusive Access to Correlator Store files
- Deploy the Correlator Store files

NOTE The following are assumed for the rest of this chapter:

1. The Developer has created the NameSpace file.
2. All references made to Correlator Store files are specified in the NameSpace file

Mutual Exclusive Access to Correlator Store files

To avoid overwriting of data in Correlator Stores due to concurrent access by multiple users (Developers or Operators) the Composer provides the facility to lock a file.

File locking for Correlator Stores functions in the following modes:

- Composer's Operator mode
- Composer's Developer mode
- Standalone Deploy script
- Correlator Store Deploy procedure. Refer to "Deploying the Correlator Store" on page 142.
- Standalone Merge script when invoked with the -namespace option. Refer to "Merging Correlator Store files" on page 114.

When a Correlator Store that is not currently in use is opened, a lock file is created. The lock file is of the format `<`*`filename`*`>.lock`, where <*filename*> is the name of the Correlator Store. Acquiring a lock provides total access on the Correlator Store file.

The creation of the lock file fails if the Correlator Store is in use. Each mode operates differently in this situation. Refer to the following table:

Mode	Action
Operator	Composer displays an error message and opens the file in read-only mode
Developer	Composer displays an error message and aborts the file open action
Standalone Deploy script	displays an error message and aborts the deploy action.
Deploy procedure	displays an error message and aborts the deploy action
Standalone merge script	displays an error message and aborts the merge action

The lock is removed when the Correlator Store is closed.

If any of the above actions abort abruptly while a Correlator Store is locked, use one of the following mechanisms to recover the Correlator Store:

- In the Operator mode, select `Options->Forcefully Unlock` after highlighting the locked Correlator Store.
  The same can be done in the case of Deploy operation from the Composer.
- In the Developer mode, the Developer can manually remove the lock file, <*`Correlator Store filename`*>`.lock` file, which resides on the same directory as the correlator store.
  The same can be done in the case of abort during standalone deploy and merge.

NOTE Though the user is allowed to make changes to the Correlator Store, the Operator does not have mutual exclusive access to this file.

WARNING **It is recommended that this option be chosen with caution as there is always a possibility that important data could be lost while multiple operators save the Correlator Store files.**

Deploying the Correlator Store

The Operator deploys the Correlator Store into the ECS engine. To load the Correlator Store file into the ECS engine, do one of the following:

- Ensure that all Correlator Stores have been saved and closed. Select `Options->Deploy`.
- Click the `Deploy` icon.

The Deploy Status window appears and displays one of the following:

- If the deploy is successful, the Deploy Status window indicates success.

- If an error occurred, the `Deploy Status` window indicates failure. To view the details of the error, select `[Details]` in the `Deploy Status` window. To close the window, select `[OK]`.

Deploy from command prompt

The Correlator Stores can be deployed from the command prompt also using the `csdeploy.ovpl` script provided. The `csdeploy.ovpl` script refers to the Deploy configuration file required by the Composer. Refer to "Step 4: Creating the Deploy Configuration file" on page 129 for details on the Deploy configuration file. The `csdeploy` script resides in the `$OV_BIN` directory. To deploy the Correlator Store, type

```
csdeploy.ovpl -p <Deploy Configuration filename>
```

where,

<*Deploy Configuration filename*> is the name of the Deploy Configuration file. If no filename is specified, the default Deploy Configuration file `$OV_CONF/ecs/CIB/deploy.conf` is selected.

The `csdeploy.ovpl -h` command summarizes the usage of `csdeploy`.

7 Composer Built-In Functions

This chapter describes all the Built-In functions provided by the Correlation Composer.

Composer Built-in Functions

The Composer comes bundled with built-in functions to perform simple logging, retrieving and manipulation of event data. The table below lists the built in functions along with their descriptions.

Table 7-1 Composer Built-in functions

Function Name	Description
add	Returns the sum of values that are passed to it
bitand	The bitwise and operation on its arguments
bitinv	The bitwise inverse of the argument
bitor	True if either argument is true
bitxor	The bitwise exclusive-or of the two arguments
div	Integer divide
getByIndex	Returns from the specified element from the list
getCounter	Returns the value stored in a counter
getHour	Returns the current hour
getMinute	Returns the current minute
getMonth	Returns the current month
getTime	Returns the time(in seconds) since epoch
makeList	Returns a list that contains the set of arguments passed to it
mod	Returns the first integer modulus the second integer
mul	Return the product of values passed to it
retrieve	Retrieves a value stored previously
retrievstr	Retrieves a string stored previously

Table 7-1 Composer Built-in functions (Continued)

Function Name	Description
`setCounter`	Stores the incremented value
`store`	Stores a value based on a key
`storeStr`	Stores the string value based on a key
`sub`	Returns the difference of the values passed to it

add

Syntax

`add` *`int1 int2`*

Where:

int1 and *`int2`* are integers.

Description

The add function returns the sum of values passed to it

Example

`add 1 2` returns the value 3

bitand

Syntax

`bitand` *`int1 int2`*

Where:

`int1` and *`int2`* are integers.

Description

The result is the integer value of a bitwise operation between the two arguments.

Example

`bitand 7 0` returns the value 0

`bitand 7 1` returns the value 1

bitinv

Syntax

`bitinv` *`int`*

Where:

int is an integer

Description The result is the integer value of a bitwise inversion of the argument. The argument is treated as a 32 bit unsigned bit pattern.

Example `bitenv 1` returns the integer -2

bitor

Syntax `bitor` *`int1 int2`*

Where:

`int1` and *`int2`* are integers.

Description The result is the integer value of a bitwise `or` operation between the two arguments. The arguments are treated as 32 bit unsigned bit patterns.

Examples `bitor 7 0` returns the value 7.

`bitor 7 1` returns the value 7.

`bitor 8 1` returns the value 9.

bitxor

Syntax `bitxor` *`int1 int2`*

Where:

`int1` and *`int2`* are integers.

Description The result is the integer value of a bitwise `exclusive or` operation between the two arguments. The arguments are treated as 32 bit unsigned bit patterns.

Examples `bitxor 7 0` returns the integer 7.

`bitxor 7 1` returns the integer 6.

`bitxor 8 1` returns the integer 9.

div

Syntax

`int1` `div` *`int2`*

Where:

`int1` is the integer dividend.

`int2` is the integer divisor.

Description

The `div` function divides *`int1`* by *`int2`* to produce an integer result.

Example

`7 div 3` results in `2`

getByIndex

Syntax

`getByIndex` *`list index failvalue`*

Where:

`list` is a list of any data types

`index` is the position from which the value is to be extracted

`failvalue` is the value returned if the `getByIndex` function fails

Description

The `getByIndex` function returns the element at *`index`* position from the *`list`* passed in. If *`index`* number of elements do not exist, then the function returns the *`failvalue`*.

Typically the *`getByIndex`* function is used to retrieve individual elements from the return value of the previous call to an external function.

Examples

Let there be a external function called `getInterfaceDetails` which returns the `interfaceName` and `interface IP Address` and this return value bound to a variable called `details`. To extract the IP address, the getByIndex function will be called as

```
getByIndex details 2 0
```

If the `getByIndex` function fails, the value returned is `0`.

See Also

- “retrieve” on page 153
- “store” on page 156

getCounter

Syntax

`getCounter` *`toInit key1, key2...`*

where,

`toInit` is the method in which the value will be retrieved.

`key1, key2,...` are the keys based on which the value is retrieved. Refer to “Concept of Keys” on page 158 to understand how the keys function.

Description

The `getCounter` function retrieves the counter values stored against the keys. The value should have been stored previously using the `setCounter` call with the same set of keys.

The field *`toInit`* can take the following values

`1`	The stored value is returned and the storage memory occupied by this value is freed.
`0`	The stored value is returned, but the storage space is not deleted and further calls to retrieve will return the stored value.

Examples

The example below illustrates the usage of the getCounter function.

`getCounter 1 agent_addr arrival_time`.

The counter value stored under the keys `agent_addr` and `arrival_time` are retrieved and the memory space occupied is freed.

See Also

- “setCounter” on page 155

getHour

Syntax

`getHour()`

Description

The `getHour` function returns the current hour. The result is an integer and value can be between 0-23. All time is represented in UTC.

getMinute

Syntax

`getMinute()`

Description The `getMinute` function returns the current minute. The result is an integer and value can be between 0-59. All time is represented in UTC.

getMonth

Syntax `getMonth()`

Description The `getMonth` function returns the current month. The result is an integer and value can be between 1-12.

getTime

Syntax *`getTime ()`*

Description The `getTime` function returns the time in seconds since epoch(1 January 1970). The result is a string.

makeList

Syntax makeList *`arguments`*

Where:

`arguments` is the list of arguments that are passed to the function

Description The `makeList` function returns a list that contains the set of arguments passed to it. Typically, this function is used as the input and/or output function in the user-defined Correlators as these functions require a list as the return type.

Examples `makeList 10, 20, 30`

mod

Syntax *`int1`* `mod` *`int2`*

Where:

`int1` and *`int2`* are both integers.

Description The result is the integer value of the remainder after dividing *`Int1`* by *`Int2`*.

Examples `7 mod 3` returns the integer `1`.

`1 mod 1` returns the integer `0`.

`7 mod (-3)`returns the integer `1`.

`(-7) mod 3`returns the integer `-1`

mul

Syntax `mul` *`int1 int2`*

Where:

`int1` and *`int2`* are two integers

Description The `mul` function returns the product of the two values.

Examples `mul 3 4` returns `12`

retrieve

Syntax `retrieve` *`toInit failvalue key1, key2,...`*

Where:

`toInit` is the method in which the value will be retrieved.

`failvalue` is the value that is returned by the function if the retrieve function fails.

`key1, key2,...` are the keys based on which the value is retrieved. Refer to “Concept of Keys” on page 158 to understand how keys function.

Description The `retrieve` function retrieves the values stored previously. It is necessary that the values to be retrieved be called under the same keys in the same order.

The parameter *`toInit`* can take the values:

`1`	The stored value is returned and the storage space occupied by this value is freed and further calls to retrieve, without a preceding call to store, will result in an error, and the `failvalue` is returned.
`0`	The stored value is returned, but the storage space is not deleted. Further calls to retrieve will return the stored value.

The parameter *`failvalue`* is the value returned if no value has been previously stored against the keys specified.

Examples

The following example illustrates the usage of the retrieve function

```
retrieve 1 0 agent_addr, Constants.Type2_SP
```

The values associated with the keys `agent_addr` and `Constants.Type2_SP` are retrieved and the memory occupied is freed. If the retrieve function fails, the value returned is `0`. All further calls to retrieve, without a preceding call to store, will result in an error and will return the failvalue.

```
retrieve 0 0 agent_addr, Constants.Type2_SP
```

The values associated with the keys `agent_addr` and `Constants.Type2_SP` are retrieved, but the memory used for storage is not deleted and all succeeding calls to retrieve will return the stored value. If the retrieve function fails the value returned is 0.

See Also

- “store” on page 156

retrieveStr

Syntax

`retrieveStr` *`toInit failvalue key1, key2,...`*

Where:

`toInit` is the method in which the value will be retrieved

`failvalue` is the value to be returned by the function if the retrieve function fails.

`key1, key2,...` are the keys based on which the value is retrieved. Refer to “Concept of Keys” on page 158 to understand how keys function.

Description

The retrieveStr function retrieves the value(as a string) stored previously via the storeStr function based on the same set of keys.

Example

The following example illustrates the usage of the retrieveStr function

```
retrieveStr 1 0 agent_addr arrival_time
```

The value associated with the keys `agent_addr` and `arrival_time` is retrieved(in string format) and the memory occupied is freed. If the retrieve function fails, the value returned is `0`.

See Also

- "storeStr" on page 157

setCounter

Syntax

`setCounter` *`toInit increment window key1, key2,...`*

Where:

`toInit` is the method in which the value will be set.

`increment` is the increment value.

`window` is the time period for which the value will be stored.

`key1, key2,...` are the keys against which the values will be set. Refer to "Concept of Keys" on page 158 to understand how keys function.

Description

The `setCounter` function increments the value stored under the keys by increment value. If no value has been previously stored then the value passed is stored. The field toInit can take the following parameters

`0`	The stored value is incriminated by the amount specified in the increment value but the storage space is not deleted and further calls to retrieve will return the stored value.
`1`	The storage memory is reintialized, freed and the value passed is returned. Note that, the return value is the stored value.
`2`	If the resultant value after the operation(increment added to the stored value) is zero the memory associated is freed.

The value is stored for as long as the time specified in the window, after which the value is backed out. For example, assume that the value in the counter before an operation is 10 and a `setCounter` operation is done, with an increment of 5 and a `window` of 3 seconds. Now, the counter value will be 15. After 3 seconds the `setCounter` operation is reversed. In this example, it would result in 5 being decremeneted from the current counter value. The `window` can also be set to 0 which means the value is stored till the keys are reintialized.

Examples

The following example illustrates the usage of the setCounter function

```
setCounter 0 6 10 agent_addr arrival_time
```

If the value stored previously is 5, the new value that will now be stored is 6+5=11. However, if there was no value stored previously, the value stored will be 6 under the keys `agent_addr` and `arrival_time`. The new value is stored for a period of 10 seconds.

See Also

- "getCounter" on page 151

store

Syntax

`store` *`value window key1, key2,...`*

Where:

`value` is the value that is to be stored.

`window` is the time period for which the value will be stored.

`key1, key2,...` are the keys based on which the value is stored. Refer to "Concept of Keys" on page 158 to understand how keys function.

Description

The `store` function stores a value based on the key(s) for a given time period or till another call to store. It is mandatory that there be at least one key for the value being stored. Another call to store under the same key(s) will overwrite the currently stored value. The parameter *`window`* can also take the value

n	The time in seconds.This value is stored for 'n' seconds.
-1	The value is stored forever.

Examples

The following example illustrates the usage of the store function

```
store uuid agent_addr, Constants.Type2_SP
```

The `uuid` specified in the event is stored under the keys `agent_addr` and `Constants.Type2_SP`

See Also

- "retrieve" on page 153

storeStr

Syntax

`storestr` *`toAppend seperator value window key1, key2,...`*

Where,

`toAppend` parameter decides how the value will be stored.

`seperator` is the field seperator.

`value` is the value to be stored.

`window` is the time period for which the value will be stored.

`key1, key2...` are the keys based on which the value will be stored. Refer to "Concept of Keys" on page 158 to understand how keys function.

Description

The `storeStr` function stores the stringified value based on the key(s) for a specified time period.

The parameter toAppend can take the following values:

`0`	The value is appended to the existing value and is stored based on the keys
`1`	The value is stored based on the keys. Any values stored previously are erased and only the new value is stored.

The parameter *`window`* can also take the value

`n`	The time in seconds.This value is stored for 'n' seconds.
-1	The value is stored forever.

Examples

The following examples illustrates the usage of the `storeStr` function

`storeStr 0 ":" Hello 10 agent_addr arrival_time`

`storeStr 0 ":" World 5 agent_addr arrival_time`

The `first call to storeStr` function stores the string `"Hello"` for a period of 10 seconds while the second call stores `World` for 5 seconds. A call to `retrieveStr` will return `Hello:World`. After 5 seconds a call to `retrieveStr` will return `Hello` and a call to `retrieveStr` after 10 seconds will return the failvalue.

See Also

- "retrieveStr" on page 154

sub

Syntax

`sub` *`int1 int2`*

Where:

`int1` and *`int2`* are the any two integers.

Description

The sub function returns the difference of values passed to it

Example

`sub 20 10` returns the integer 10

Concept of Keys

For all the `store` and `retrieve` functions (that is, `store`, `retrieve`, `storeStr`, `retrieveStr`) the value stored/retrieved is against the keys passed into the function as parameters. The functions expect a minimum of one key to be passed in. However, multiple keys can also be used. When multiple keys are used the function internally would concatenate the values referred to by these keys and create a single key.

For example, a key X which holds a value abc is equivalent to the set of keys `x, y, z` where they hold values `a, b, c` respectively. Ensure the order of the keys is maintained. Taking the above example, passing in keys `z, y, x` would result in a final key value of cba and NOT abc.

The `store` and `retrieve` functions use a global hash table. While this is a powerful mechanism of passing data between Correlators, an incorrect usage would result in Correlators overwriting each others spaces. For example, consider Correlator1 stores a value against a key whose value is abc, and Correlator2 stores a value against a key(s) whose value also evaluates to abc. In such a situation, the value stored would be the last value stored. To ensure that Correlators do not step on each other, keys should be chosen such that they are unique. (a good way to ensure this is to use the Correlator Name as part of the key)

NOTE

The `store` and `retrieve` functions use a different hash table than that of a `storeStr` and `retrieveStr`.

8 Use Cases

This chapter provides use cases to help you understand the workflow schema defined in the Correlation Composer. You can plan your system configuration following the examples in this chapter. Use cases are provided to define the following Correlator Templates:

- "Case 1: Enhance Correlation" on page 161
- "Case 2: Multi-Source Correlation" on page 164
- "Case 3: Rate Correlation" on page 168
- "Case 4: Repeated Correlation" on page 172
- "Case 5: Suppress Correlation" on page 176
- "Case 6: Transient Correlation" on page 179
- "Case 7: Multi Event Correlation accessing external topology" on page 185

NOTE All the examples of alarms used in this document used the SNMP Trap-PDU format. The Composer however is format independent and supports CMIP, OPC and X733 event types.

Case 1: Enhance Correlation

Consider a Temperature alarm as shown below.

```
Trap-PDU
enterprise {1 2 3 4 995},
agent-addr internet : “\x0A\x00\x01\x7F”
generic-trap 6,
specific-trap 95,
time-stamp 414746291,

variable-bindings{

    {

       name { 1 3 6 1 4 1 11 2 7 2 17 0},
      value simple : number : 95

    }

   }
}
```

Looking at the alarm, it is not immediately evident to the operator what the problem could be. The requirement is to add a variable binding with the string “Temperature of the device is too high - Please check for air-conditioning and/or fan failure”

What you need to know?

1. **How do you identify Temperature alarms?**

   All Temperature alert failure will have the following attributes which will identify them

   - `enterprise` is set to 1.2.3.4.995
   - `generic-trap` is set to 6
   - `specific-trap` is 95

2. **How do you differentiate a Temperature ON alarm from an Temperature OFF alarm?**

   - if the `specific-trap` is set to 95, it is an Temp ON alert

3. **What do you do with the original alarm?**

   The user can choose to retain the original alarm along with the enhanced alarm. The table below describes the functionality of the buttons in the Enhance Correlator Template window.

Button Name	**Selected**	**Functionality**
`Want Original`	Yes	The original alarm is output along with the enhanced alarm
	No	Only the enhanced alarm is output, the original alarm is discarded.
`Enhance Always`	Yes	Alarms will be modified and output regardless of any other correlation deciding to discard this alarm
	No	Alarms will be modified if no other Correlator decides to discard this alarm

NOTE It is recommended that the `[Enhance Always]` button be selected with caution, as this leads to Enhancing all alarms.

Follow the procedure given below to define the Enhance Correlator Template:

1. Select `Correlations:Correlator Templates->Enhance` from the Correlator Store window. The Enhance Correlator Template window opens.
2. Enter the `Name` for the Correlator in the Name text box.
3. Enter the Description for the Correlator in the `Description` text box of the Correlator window.
4. Enter the following values to define the Alarm Signature
   - `enterprise = 1.2.3.4.995`
   - `generic-trap = 6`
   - `specific-trap = 95`

5. Declare the variable `errstr`

   a. Type `errstr` in the Name cell.

   b. Select “Constant” from the Operator drop down menu

   c. Enter the string in the Value field “Temperature of the device is too high - Please check for air-conditioning and/or fan failure”

NOTE The above steps will hereinafter be referred as `errstr constant <str>` or the equivalent of the above expression.

6. Click on the `New Alarms` tab to alter the alarm. The New Alarm panel opens.
7. Select `Alter Specification` from the pop down menu. The `Alter Alarm Definition` table is displayed.
8. Define the following attributes to alter the alarm
   - select `variable-bindings[1].value` from the `Field` drop down menu
   - select `replace` from the Mode drop down menu
   - select `errstr` from the Value pop up menu
9. Click on `[OK]` to complete the definition of the Correlator.

   Notice that the correlation you have just defined is displayed in the Correlator Store table.

Case 2: Multi-Source Correlation

Between two switching entities there exist multiple redundant SS7 links, which together form a logical entity called a signalling set. If the trunk between the two fails then an SS7 Link Set failure is received with a SS7 failure alarm for each individual SS7 link. The requirement is to suppress all individual SS7 failures and forward only the SS7 Link Set failure.

A sample SNMP trap PDU for an SS7 Link failure could appear in an event log as below:

```
Trap-PDU{
    enterprise{1 2 3 4 997}
    agent-addr internet ; “\x0A\x00\x01\x7F”,
    generic-trap 6,
    specific trap 55,
    time-stamp 414746291,
    variable-bindings {
     {
      name {1 3 6 1 4 1 11 2 17 17 0},
      value simple : string “Link Failure -10 on Signalling set 2”
     }
  }
}
```

A sample SNMP trap PDU for an SS7 Link Set failure could appear in an event log as below:

```
Trap-PDU{
  enterprise {1 3 6 1 4 1 999 9}
  agent-addr internet : “\x0A\x00\x01\x7F”,
  generic-trap 6,
  specific-trap 56,
  time-stamp 414746291,
  variable-bindings {
      name { 1 3 6 1 4 1 11 2 17 2 17 0},
     value simple :string :”Link Set Failure - 2”
    }
  }
}
```

What you need to know?

1. **How do you identify the SS7 Link failure alarms?**

   All SS7 Link failure alarms will have the following attributes that can identify them:

   - `enterprise` set to 1.2.3.4.997
   - `generic-trap` set to 6
   - `specific-trap` set to 55

2. **How do you identify the SS7 Link Set failure alarms?**

   All SS7 Link Set Failure alarms will have the following attributes that can identify them:

   - `enterprise` set to 1.2.3.4.999
   - `generic-trap` set to 6
   - `specific-trap` set to 56

3. **How do you identify the SS7 Link and SS7 Link Set failures are emitted from the same device and belong to the same set?**

   - If the SS7 Link Set ID and the SS7 Link ID are the same AND the agent addresses for both the failure alarms are the same, then the SS7 Link failure belongs to the SS7 Link Set failure.

4. **What do you with the various alarms?**

   Alarms can be discarded based on whether the set is complete or not. The table below describes the functionality of the various buttons in the Multi-Source Correlator Template window.

Button Name	Selected	Functionality
`Discard on Set Completion`	Yes	The alarm will be discarded if the set is complete, else will be forwarded
	No	The alarm will be forwarded regardless of set completion.

Button Name	Selected	Functionality
`Window Period`		Mandatory Field - The time period within which all alarms of the set need to arrive for the set to be considered complete. The alarms can arrive in any order.
`Set`	No	Operates in Mode 1. Refer to "Multi-Source Correlator Template" on page 25
	Yes	Operates in Mode 2. Refer to "Multi-Source Correlator Template" on page 25

Follow the procedure given below to define the Multi-Source Correlator Template:

1. Select `Correlations->Correlation Templates->Multi-Source` from the Correlator Store window. The Multi-Source Correlator opens.
2. Enter the `Name` of the Correlator and `Description` of the Correlator.
3. Enter the Description of the Correlator in the `Description` text box.
4. Click on the Definition tab to open the `Alarm Definition` panel. Name the alarm. In the `Name` panel on the left side of the window, enter the name for the alarm-SS7 Link failure.
5. Define the `Alarm Signature` to identify the SS7 Link failure alarms

   Enter the following values in the `Alarm Signature` table

   - `enterprise id`=1.2.3.4.997
   - `generic trap`=6
   - `specific trap=55`
6. Declare the following variables in the `Variables` table

   - SS7 Link ID extracted from `variable-bindings[0].value`. Enter the name of the variable- SS7 Link ID

- Type set to "Extract". Enter *#-#<#.linkID>* in the `Extract Pattern` window.
- Pattern Separator set to "#"

This extracts the SS7 Link ID and assigns the extracted numeral to the variable `linkID`

7. Define the `Message Key`. Click on the Message Key text box. A pop up menu is displayed. Select SS7 Link Failure->SS7 Link ID->linkID.
8. To create a set of the alarms, click the `Set` check box.
9. If you want to alter the alarm, define the changes in the `Alter Alarm Definition` table.
10. Repeat Steps 3 to 7 to define more alarms. Right click and select Add to add a new alarm. In this case
    - Define the Alarm Signature to identify the SS7 Link Set failure
      - `enterprise`= 1.3.6.1.4.1.999.9
      - `generic-trap` =6
      - `specific-trap`=56
    - Declare a variable SS7 Link Set failure extracted from `Variable Bindings[0].value.` Enter the extract pattern = `*<S><#.setID>`
    - Select the `Message Key` to `SS7 Link Set Failure->SS7 Link Set ID->setID`.
11. Enter the following parameters in the `Parameters` Panel:
    - Define the Window Period for which you want to monitor the occurrences of the alarms.
    - Select the Set Complete check box to emit the alarm only if the set is complete. For example a Power Down alarm arriving after an occurrence of a Power Up and Power Down pair will not be emitted out until a Power ON alert is received.
12. Click on `[OK]` to complete the definition of the Signature file. Notice that the correlation you have just defined is displayed in the Correlator Store table.

Case 3: Rate Correlation

Radio antennas frequently report failures during bad weather conditions. The requirement is to discard all radio antenna failures if the rate of failure is below the 5 failures in 30 minutes. If the rate exceeds this threshold then forward the alarm to the browser after annotating the alarm with the rate.

A sample SNMP trap PDU for an Antenna failure could appear in an event log as below:

```
Trap-PDU{
  enterprise {1 2 3 4 998}
  agent-addr internet : “\x0A\x00\x01\x7F”,
  generic-trap 6,
  specific trap 80,
  time-stam 414746291,
  variable-bindings{
   {
     name{1 3 6 1 4 11 2 17 2 1 0},
     value simple:number : 2

   },
  {
     name { 1 3 6 1 4 11 2 17 2 2 0},
     value simple : string : “Ant#10#BTS#20”
  }
 }
}
```

What you need to know?

1. **How do you identify the alarms for which the count will be maintained?**

   A count will be maintained for alarms whose attributes have the following values

   - `enterprise` is 1.2.3.4.998
   - `generic-trap` is 6
   - `specific-trap` is either 80
   - `variable-bindings[0].value` is 2

2. **What do you with the alarms?**

   Duplicate alarms can be discarded. However, the correlation has to be defined to monitor the time the alarm is discarded or output. The table below describes the functionality of the buttons in the Rate Correlator Template window.

Button	Selected	Functionality
Window Period	N/A	Mandatory Field - Time period for which the alarm arrival rate is monitored
Count	N/A	Mandatory Field - The threshold count. If the number of alarms exceeds threshold count within the specified Window Period then the rate threshold is considered breached.
Discard	Yes	All alarms are discarded. Only the new alarm, if created, is output.
	No	Alarms are not discarded. Additionally the new alarm, if created, is output.

Follow the procedure given below to define the Rate Correlator Template:

1. Select `Correlations:Correlator Templates->Rate` from the Correlator Store. The Rate Correlator Template window opens.
2. Enter the `Name` and `Description` for the Correlator.
3. Click on the `Definition` tab to display the Alarm Definition panel. Enter the following values to identify the Alarm Signature
   - `enterprise`
   - `generic-trap`
   - `specific-trap`
4. Declare the following variables
   - `ant` - This is a variable that in combination of the extracted pattern will specify the Antenna ID and BTS ID.

- Extract the Antenna ID and BTS from `variable-bindings[1].value`. In the extract pattern window enter `Ant<S><*.antid><S>Bts<*.btsid>`
- In the Pattern Separator field, enter #

Two alarms are emitted from the same Antenna and BTS if their corresponding `antid`, `btsid` and `agent-addr` are the same.

- `mkey` - This is the unique field that will combine all the above attributes into one and constitute the `Message Key`.
  - Combine the attributes `ant.antid`, `ant.btsid`, `agent-addr`

5. Select the Message Key. Click in the MessageKey window. A pop up menu displays all attributes and pre-defined variables. Select `mkey` from the menu.
6. Define the parameters for the correlation
   - `Window Period` = 30 minutes
   - `Count` = 5
7. Select the `Discard` button if the alarms are to be discarded. Though the alarms are discarded, the count of alarm arrival is maintained.
8. Before a new alarm is created, it is necessary to define the error string that declares the problem. Define the following variables in the Variable table
   - `str1 constant "The threshold has been breached for the antenna "`
   - `str2 constant "from BTS"`
   - `errstr combine of str1, ant.antid, str2, ant.btsid`

   The above definition creates an errstr which will look like `"The threshold has been breached for the antenna 10 from BTS 20"`
9. Define the new alarm. Click on the `New Alarms` tab to alter the alarm. The `New Alarm` panel opens.
10. Select `New Alarm Specification` from the drop down menu. The `New Alarm Definition` table is displayed.
11. Select the following to define the change

- `enterprise = enterprise`
- `agent-addr = agent-addr`
- `generic-trap = generic-trap`
- `specific-trap = specific-trap`
- `time-stamp = time-stamp`
- `varBind[0]->name=varBind[0]->name`
- `varBind[1]->value = errstr`

12. Click on `[OK]` to complete the definition of the Correlator. Notice that the Correlator you have just defined is displayed in the Correlator Store table.

Case 4: Repeated Correlation

In general terms, duplicate alarms are messages that report the same alarm. You can use Repeated Correlation to suppress duplicate messages based on a variety of suppression types. Repeated Correlation is used to monitor duplicate alarms arriving within the specified Window Period.

Routers generate a CPU-Hog alarm when the utilization exceeds the threshold. The requirement is to pass only the first alarm for a given router in a 30 minute time window and discard all other alarms received in the same window. Additionally, at the end of the 30 minute period a new alarm must be generated indicating the number of such alarms received(and discarded)

A sample trap PDU could appear in a log as

```
Trap PDU{
  enterprise {1 2 3 4 6},
  agent-addr internet:”\x0A\x00\x01\x7F”,
  generic-trap 6,
  specific-trap 25,
  time-stamp 41474291,
  variable-bindings { }

}
```

What you need to know?

1. **How do you identify which alarms are duplicate?**

   All alarms with the following attributes are identified as Duplicate alarms.

   - `enterprise` is set to 1.2.3.4.6
   - `generic-trap` is set to 6
   - `specific-trap` is set to 25

2. **How do identify that the alarms are emitted from the same router?**

   Two alarms are said to be coming from the same router if the agent address of the router from which they are emitted is the same.

3. **What do you want to do with the duplicate alarms?**

   Duplicate alarms can enter one of the following states

   - Discarded

     The event are discarded from ECS and are not available for further correlation.

   - Output

     If the event is to be output, it should be further decided if this event should take part in other correlations.

4. **When should the alarms be discarded or output?**

   You can choose to discard or output alarms whenever required, based on the Correlator definition. The table below describes the functionality of the buttons in the Repeated Correlator Template window.

Button	**Selected**	**Functionality**
`Window Period`		Mandatory field - Time period for which the alarm duplication is monitored.
`Discard Duplicate`	Yes	Chooses Mode 1 of operation. Refer to “Repeated Correlator Template” on page 27
	No	Chooses Mode 2 of operation. Refer to “Repeated Correlator Template” on page 27
`Discard Immediately`	Yes	This is applicable only if the Discard Duplicate button is chosen. The effect of this is that all duplicate alarms will be discarded without participating in other correlations.
	No	Duplicate alarms will be discarded only after participating in other Correlators.

Follow the procedure given below to define the Repeated Correlator Template:

1. Select `Correlations->Correlator Templates->Repeated` from the Correlator Store window. The Repeated Correlator Template window is displayed.
2. Enter the `Name` and `Description` for the Correlator.
3. Click on the `Definition` tab to display the Alarm Definition panel. Enter the following values to define the Alarm Signature
   - `enterprise` = 1.2.3.4.6
   - `generic-trap` = 6
   - `specific-trap = 25`
4. Declare the following variables
   - `str constant " alarms discarded in 30 minutes"`
   - `errstr combine of AlarmCnt and str`
5. Select the `Message Key = agent-addr`
6. Define the Window period for which the alarm arrival must be maintained. Enter 30 minutes in the Window Period field.
7. Check the[`Discard Duplicate]` and [`Discard Immediately]` buttons, as duplicate alarms are to be discarded as soon as they have taken part in this correlation and you do not wish it to take part in other correlations.
8. Define the new alarm. Click on the `New Alarms` tab. The New Alarm panel is displayed.
9. Select `New Alarm specification` from the drop down menu. The Create Alarm Definition table is displayed.
10. The table lists the mandatory fields to be filled for an event. Set the following values:
    - `enterprise = enterprise`
    - `agent-addr = agent-addr`
    - `generic-trap = generic-trap`
    - `specific-trap = specific-trap`

- `time-stamp = time-stamp`
- `varBind[1]->value = errstr`

11. Click on `[OK]` to complete the definition of the Correlator. Notice that the Correlator you have just defined is displayed in the Correlator Store table.

Case 5: Suppress Correlation

Movement traps in general need investigation, however, if the movement alarms are from exchanges emitted from the City offices, they can be discarded as there is always movement and the alarm browser is filled with these alarms. The requirement is to discard all movement alarms emitted from the City offices.

A sample SNMP trap PD could appear in a log as

```
  Trap PDU {
  enterprise{1 2 3 4 999},
  agent-addr internet : “\x0A\x00\x01\x7F”,
  generic-trap 6,
  specific-trap 1,
  time-stamp 414746291,
  variable-bindings{
         {
           name {1 3 6 1 4 1 11 2 17 2 1 0},
           value simple : number 2
         },
          {
           name {13 6 1 4 1 11 2 17 2 2 0},
           value simple : “City-Bangalore”
          }
          {
          name {1 3 6 1 4 11 2 17 2 17 0},
          value simple : string : “There is movement”
         }
    }
}
```

What you need to know?

1. **How do you identify which alarms are to be Suppressed?**

   All movements will have the following attributes which will identify them

   - `enterprise id` is set to 1.2.3.4.999
   - `generic trap` is set to 6
   - `specific trap` is set to 1

- `variable bindings[1].value` will have the string “City”
- `variable bindings[2].value` will have the string “There is Movement”.

2. **What are the parameters to be configured and what do they mean?**

   The table below describes the functionality of the different buttons in the Suppress Correlator Template window.

Button Name	**Selected**[a]	**Functionality**
Participate In Other Correlation	No	The alarm does not participate in other correlations before it is discarded.
	Yes	The alarm takes part in other correlations, before it is discarded.

a. Indicates if the button is chosen by the user. The effect of selecting it or not selecting it, is described in the Functionality column.

Follow the procedure given below to define the Suppress Correlator Template:

1. Select `Correlations:Correlator Templates->Suppress` from the Correlator Store window. The Suppress Correlator Template window opens.
2. Enter the name of the Correlator in the `Name` text field.
3. Enter the description of the Correlator. The `Description` can briefly state what the Correlator is expected to do.
4. Click on the `Definition` tab to display the Alarm Definition panel.

   Set the following values to define the Alarm Signature

   - `enterprise` = `1.2.3.4.999`
   - `generic-trap` = `6`
   - `specific-trap` = `1`
   - `variable-bindings [2].value` = `“There is movement”`
   - `variable-bindings [1].value` `matches “City”`

5. Click [OK] to complete the Correlator. Notice that the correlation you have just defined is displayed in the Correlator Store table.

Case 6: Transient Correlation

PCM Links go out of synch frequently and a trap is generated indicating a link failure. However, the two ends of the PCM link typically re-synch within 1 second. The requirement is to hold the PCM Down alarm for a period of 2 seconds and discard it if the PCM Up alarm is received within this period. An additional requirement is that if 5 such link failures are detected in a 30 minute period then a new alarm needs to be generated indicating instability. The newly created alarm should indicate the time taken for the breach to take place.

A sample trap PDU would appear in a log as:

```
Trap-PDU {
   enterprise {1 2 3 4 999}
   agent-addr internet : “\x0A\x00\x01\x7F”,
   genric-trap 6,
   specific-trap 400,
   time-stamp 414746291,
   variable-bindings{
    {

      name {1 3 6 1 4 11 2 17 2 1 0},
     value simple : number : 400

    }
  }
}

Trap-PDU {
   enterprise {1 2 3 4 999}
   agent-addr internet : “\x0A\x00\x01\x7F”,
   genric-trap 6,
   specific-trap 400,
   time-stamp 414746291,
   variable-bindings{
    {

      name {1 3 6 1 4 11 2 17 2 1 0},
     value simple : number : 401

    }
  }
}
```

What you need to know?

1. **How do you identify PCM trap alarms?**

   All PCM link alarms will have the following attributes which will identify them

   - `enterprise` is set to 1.2.3.4.999
   - `generic-trap` is set to 6
   - `specific-trap` is contained in [400, 401]

2. **How do you differentiate a PCM clear alarm from a PCM link failure alarm?**

   - if the `specific-trap` is set to 400, it is a PCM link failure alarm
   - if the `specific-trap` is set to 401, it is a PCM clear alarm

3. **How do you know the alarms are coming from the same PCM link?**

   - `variable-bindings[0].value` contains the PCM Link-ID
   - if two alarms have the same `agent-addr` and same PCM Link-ID, then they are coming from the PCM link

4. **How is the time maintained?**

   It should be further decided what the time interval for correlation to be monitored is and the time period for which a failure alarm waits for a Clear alarm to arrive. The table below describes the functionality of the various buttons in the Transient Correlator Template window.

Button Name	Selected	Functionality
`Window Period`		Mandatory Field - The maximum time a Failure alarm is held by the Composer waiting for a Clear Alarm. If the Clear alarm is received while the alarm is held, both the Clear and Failure alarms are discarded. If no Clear alarm is received in this Window Period, the failure alarm is forwarded. Typical hold periods are between 1-10 minutes depending on the severity of the problem.
`Enable Threshold`	Yes	Maintains a count of the number of alarm pairs for the specified Threshold Window. If the count equals the Threshold Count within the Threshold Window, a new alarm is created and forwarded.
	No	No Count is maintained. The Threshold Count and Threshold Windows are both disabled.
`Threshold Count`		This button is enabled only when the `Enable Threshold` button has been enabled. The number of alarm pairs, viz the Failure and Clear alarms arriving
`Threshold Window`		This button is enabled only when the `Enable Threshold` button has been enabled. The time period for which the count is maintained

Follow the procedure given below to define the Transient Correlator Template:

1. Select `Correlations:Correlator Templates->Transient` from the Correlator Store window. The Transient Correlator Template window is displayed.
2. Enter the `Name` and `Description` for the Correlator.
3. Click on the `Definition` tab to display the Alarm Definition panel. Enter the following values to identify the Alarm Signature
   - `enterprise = 1.2.3.4.999`
   - `generic-trap` = 6
   - `specific-trap is in list [400, 401]`

NOTE The Alarm Signature must be defined such that both the Clear and Failure alarms will pass this condition.

4. Declare the following variables
   - `clear constant 401` to indicate what is the attribute of the clear alarm that will differentiate it from the failure alarm.
5. Define the time period within which a clear alarm must arrive after the failure alarm has arrived. In the `Window Period`, enter 2 seconds.
6. Define the clear alarm. Click on the `Clear Alarm` button. The `Clear Alarm` window opens. Select the following from the drop up menu under the respective headings
   - `Attribute = Specific-trap`
   - `Operator = '='`
   - `Value = clear`
7. Define the `Message Key`. The Message Key is a combination of the PCM Link-ID and the agent address(this is to identify uniquely the PCMLink from which these alarms are emitted). A variable has to be defined that contains this value. To declare this variable:

   a. In the Variables table, enter PCMLink in the Name cell.

   b. Select the `Operator = Combine`. The `Combine Definition` window is displayed.

c. Select the parameters `agent-addr` and `variable-bindings[0]->value` from the pop up menu.

d. Close the `Combine Definition` window.

This defines the `Message Key`.

8. Select PCMLink from the Message Key pop up menu.
9. Define the new alarm that will be output. Prior to this some variables containing the error string must be created. Declare the following variables:

   a. `errstr1 constant “Threshold breached in ”`

   b. `errstr2 constant “seconds”`

   c. `errstr = combine of str1, CorrelationDuration, str2`

   `CorrelationDuration` is an variable automatically generated by the Composer to monitor the time taken for the threshold to be breached.

10. Define the Threshold Window period. Enter the following values
    - Click in the Enable Threshold checkbox.
    - Threshold Count = 5
    - Threshold Window = 30 minutes
11. Define the new alarm that will be output. Click on the New Alarm tab. The New Alarm panel is displayed.
12. Select New Alarm Definition from the drop down menu. The New Alarm Definition table is displayed. Select the following values:
    - `enterprise = enterprise`
    - `agent-addr = agent-addr`
    - `generic-trap = generic-trap`
    - `specific-trap = specific-trap`
    - `time-stamp = time-stamp`
    - `varBind[0]->name=varBind[0]->name`
    - `varBind[1]->value = errstr`
13. Click on `[OK]` to complete the definition of the Correlator.

Notice that the correlation you have just defined is displayed in the Correlator Store table.

Case 7: Multi Event Correlation accessing external topology

In a network, if an MSC fails and a BSC connected to the failed MSC also fails within 5 minutes, then add a valid message to the BSC indicating what the problem may be.

A sample SNMP trap PDU for an MSC Failure could appear in an event log as below:

```
TrapPDU {
   enterprise { 1 2 3 4 996},
   agent-addr internet : “\x0A\x00\x01\x7F”,
   generic-trap 6,
   specific-trap 65,
   time-stamp 414746291,
   variable-bindings {

   {
     name {1 3 6 1 4 1 11 2 17 2 17 0},
     value simple : string :”MSC Failure-MSC_ID=MSC1”
   }
 }
}
```

A sample SNMP trap PDU for an BSC Failure could appear in an event log as below:

```
Trap PDU {
   enterprise { 1 2 3 4 995},
   agent-addr internet : “\x0A\x00\x01\x7F”,
   generic-trap 6,
   specific-trap 66,
   time-stamp 414746291,
   variable-bindings {
      {
        name {1 3 6 1 4 1 11 2 17 2 17 0},
       value simple : string :”BSC Failure-BSC_ID=BSC1”
      }
   }
}
```

What you need to know?

1. **How do you identify an MSC failure alarm?**

   All MSC failures will have the following attributes that can identify them

   - `enterprise` is 1.2.3.4.996
   - `generic-trap` is 6
   - `specific-trap` is 65

2. **How do you identify an BSC failure alarm?**

   All BSC failures will have the following attributes that can identify them

   - `enterprise` is 1.2.3.4.995
   - `generic-trap` is 6
   - `specific-trap` is 66

3. **How do you know the MSC and BSC are connected?**

   Studying the alarms alone does not help identify how the alarms are related to each other. An external application has to be invoked to be able to understand the topology. A user defined function is written so that it can extract the required information.

   For example, in this case, a function `getname()` is defined in such a way that it takes the BSC ID as a parameter and returns the name of the MSC to which it is connected. If the value returned is the same as the MSC that emitted the failure alamr, then the two devices are connected.

Follow the procedure given below to define the Multi-Source Correlator Template:

1. Select `Correlations->Correlator Templates->Multi-Source` from the Correlator Store window. The Multi-Source Correlator Template window opens.
2. Enter the `Name` and `Description` of the Correlator.
3. Click on the `Definition` tab to display the Alarm Definition panel.

Name the alarm. In the `Name` panel on the left side of the window, enter the name for the alarm - `MSC Failure`.

4. Declare the `Alarm Signature` to identify the MSC Failure alarms

   Enter the following values in the Definition table

   - `enterprise` = 1.2.3.4.996
   - `generic-trap` = 6
   - `specific-trap = 65`

5. Define the variable - `MSCLinkID`

   - MSC Link ID extracted from `variable-bindings[0].value`. Enter the name of the variable- `MSC Link ID`
   - Type set to “Extract”. Type `*#-#<#.mscID>*`
   - Pattern Separator set to “#”

   This extracts the MSC Link ID and binds it to the variable `mscID`.

6. Define the Message Key. Click on the Message Key text box. A pop up menu is displayed. Select `MSC Failure->MSCLinkID->mscID.`

7. If you would like to alter the alarm, define the changes in the Alter Alarm Definition table.

8. To add a new alarm, right click the mouse button in the Name panel on the left and enter `BSC Failure.` In this case

   - Define the Alarm Signature to identify the `BSC Failure`
     — `enterprise` = 1.2.3.4.995
     — `generic-trap` = 6
     — `specific-trap` = 66
   - Declare a variable `BSCLinkID` extracted from `variable-bindings[0].value`.
   - Select the Message Key to `BSC Failure->MSCName`.

9. To be able to identify how the MSC and BSC are connected, define a function (say `getname()`)that will perform the required operation. In this case, you can define a function `getname()` that takes the BSC name as parameter and returns the name of the MSC to which it is connected.

10. Declare the following variables:
    - `str1` constant “MSC”
    - `str2 constant “has reported a failure. The BSC failure us probably a result of this”`
    - `errstr` combine `str1, mscID, str2`
11. Enter the following parameters in the Parameters Panel:
    - Define the `Window Period` for which you want to monitor the occurrences of the alarms.
    - Select the `Set` check box to emit the alarm only if the set is complete.
12. Define the new alarm. Click on the New Alarm tab. The New Alarm panel is displayed.
13. Select New Alarm Definition from the drop down menu. The New Alarm Definition is displayed. Select the following values:
    - `enterprise = enterprise`
    - `agent-addr = agent-addr`
    - `generic-trap = generic-trap`
    - `specific-trap = specific-trap`
    - `time-stamp = time-stamp`
    - `varBind[0]->name=varBind[0]->name`
    - `varBind[1]->value = errstr`
14. Click on `[OK]` to complete the definition of the Correlator.

    Notice that the Correlator you have just defined is displayed in the Correlator Store table.

9 Troubleshooting the Composer During Runtime

This chapter provides directions for troubleshooting the Correlation Composer during runtime.

The following sources of information are useful to review before beginning to debug problems.

Release Notes

The ECS Release Notes describe late changes in the software and can help you resolve common faults. Refer to the Release Notes under `$OV_RELNOTES` on HP-UX, Solaris and Linux and%OV_RELNOTES% on Windows.

Troubleshooting the Composer

The Correlation Composer provides the facility to enable tracing to follow program and data flow. Tracing helps to debug a problem at a faster speed. HP's Cross Platform Library(XPL) has been used to perform tracing. To enable tracing for the Correlation Composer, while starting the Composer include the `-Debug` option. Type,

```
ovcomposer -debug
```

The HP OpenView Tracing subsystem comes with a variety of tools for controlling and monitoring trace messages. The most important components to be used while tracing messages in the Composer are listed below along with a brief description:

- Trace Server - The Trace Server is a process that provides an interface between trace-enabled applications and tools wanting to configure the tracing in those applications or monitor their trace output
- Trace Configuration file - The Trace Configuration file contains the tracing specifications like the application name being traced, location of trace files, the kind of messages that you want to trace. Trace configuration files are ASCII text files that can be viewed or modified using a standard text editor
- Trace Monitor - A program, such as `tracemon`, or `trcmon` that receives messages forwarded by the default Trace Server and then either displays them interactively or archives them to disk

NOTE It is assumed that the user is familiar with HP OpenView's XPL Tracing methodology. For more information refer to the XPL documents.

Trace Configuration File

When an application initializes its trace configuration, it can get the initial trace configuration information from a variety of places. There is a well-defined order as to where the configuration code looks for the application's trace configuration data.

The Trace Configuration file is available at the following locations

on HP-UX

`$OV_CONF/ecs/CO/OVCompTrc.tcf`

on Linux

`$OV_CONF/ecs/CO/OVCompTrc.tcf`

on Solaris

`$OV_CONF/ecs/CO/OVCompTrc.tcf`

on Windows

`%OV_CONF%\ecs\CO\OVCompTrc.tcf`

Following is a sample Trace Configuration file

```
TCF Version 3.2
APP: "ECSComposer"
SINK: File "c:\\openview\\log\\composer.trc" / "flush=1;maxfiles=10;maxsize=10;"
TRACE: "ECSTrace" "Event" Info Warn Error Developer Verbose / Location Stack
```

Ensure that the Trace Configuration file is present in the current directory if static tracing is chosen. Also, follow the standards, like single space, while editing the Trace Configuration file.

Basic structure

The Trace Configuration file has the following fields that must be filled

- TCF Version 3.2 - The version of the Trace Configuration file.
- APP - The application that is traced. In this case the Composer.
- SINK - The location of the tracing component, specifications of the trace file. This location can be overridden.
- TRACE - The first argument is the trace component name and it should be in double quotes. The second argument is the trace category name and it should also be in double quotes. If you are using one of the standard categories in the code, then it will be mapped to a string value (which you supply here).

The trace file is in binary format and can be viewed using the Trace Monitor utility. To view the binary file, start the Trace Monitor. Type

`run trcmon` on HPUX

`run tracemon` on Windows.
The Trace Monitor on Windows is a GUI.

Tracing in the NNM environment

Follow the procedure given below to enable tracing in the NNM environment:

1. Check if the TraceServer is running. If it is not already running, start the TraceServer. Refer to the XPL documents for more details.
2. Edit the Trace Configuration file. Following is a sample of the Trace Configuration file on HP-UX

   ```
   TCF Version 3.2
   APP: "ECSComposer"
   SINK: Socket "<server name>" ""
   TRACE: "ECSTrace" "Event" Info Warn Error Developer Verbose
   ```

   Following is a sample of the Trace Configuration file on Windows

   ```
   TCF Version 3.2
   APP: "ECSComposer"
   SINK: Socket "<server name>" "node=<ip-address>;”
   TRACE: "ECSTrace" "Event" Info Warn Error Developer Verbose
   ```

   Contrast this with the Trace Configuration file discussed earlier where the location of the trace file is specified. In this case, the name of the Trace Server should be included in this file. The Trace Configuration file is available at the following locations

   on HP-UX

   `$OV_CONF/ecs/CO/OVCompTrc_App.tcf`

   on Linux

   `$OV_CONF/ecs/CO/OVCompTrc_App.tcf`

   on Windows

   `%OV_CONF%\ecs\CO\OVCompTrc_App.tcf`
3. Associate the Trace Configuration file with the application that is being traced. Type

   `trccfg -server <server-name> <configuration filename>`
4. Monitor the trace output and view the trace output file created. To do this, the Trace Monitor utility must be running. Type

`run trcmon` on HP-UX and Linux

`run tracemon` on Windows.
The Trace Monitor on Windows is a GUI.

5. Start the Correlation Composer.

Troubleshooting the Composer during Runtime

Event Flow

A quick refresher on the event flow through the Composer would be helpful before going into how to debug the Correlators.

An event that flows through the Composer circuit has the following phases

1. On entering the circuit, the event is evaluated against the Alarm Signature for all the Correlators. If the event passes none of them, the event is output
2. The event is evaluated against the Advanced Filters for all the Correlators for which the Alarm Signatures matched.
3. The logic for the Correlators for which the event passed both the filters is executed, one after the other. Each Correlator returns what needs to be done to the event.
4. All the actions of the individual Correlators that processed together and the final outcome is dependent on the Correlators specified. If the event is to be held then the event is forwarded to the event hold mechanism.

NOTE Holding an event refers to all the three kinds of holds that is, `Hold`, `WeakHold` and `PseudoHold`.

In the case of `Weak` and `PseudoHold`, different Correlators may ask for different hold periods and the output logic of each of the Correlators are executed at the requested times.

In the case of `Hold`, if multiple Correlators ask for `Hold`, then the event is held for the longest duration requested.

5. If an event is held then it comes out after the period specified and all Correlators that asked for the event to be held is executed. Each Correlator returns what needs to be done to the event

6. All the actions of the individual Correlators are processed together and the final outcome is dependent on the Correlators specified.

Format of the Trace Message

The format of the trace file is as follows:

```
TRACE [interpreter]: Composer : <time stamp>.000000Z : <Event Id> : <Correlator
Name> : <Informative Message>
```

The trace file has 6 fields separated by a ‘`:`’ (colon), where

- The first field is always `TRACE [interpreter]`
- The second field is always `"Composer"`
- The third field is a time stamp of the format yyyymmddhhmmss followed by the time in milliseconds (which is always 000000Z)
- The fourth field is the `eventId` of the event being processed
- The fifth is either the name of the Correlator that is being processed or "Common" if the trace message is part of the common processing of events.
- The sixth field is the trace message.

Setup

Enable tracing

The following steps explain how to enable tracing on the system.

1. Turn on tracing in the Composer using the following command:

   - In NNM

     For UNIX

     ```
     ecsmgr -fact_update Composer \
     $OV_CONTRIB/ecs/CO/CompTraceOn.fs
     ```

     For Windows if Composer is installed on the C drive

     ```
     ecsmgr -fact_update Composer \
     %OV_CONTRIB%\ecs\CO\CompTraceOn.fs
     ```

2. Turn tracing on for ECS

   - For NNM

     a. `ecsmgr -i 1 -trace 65536`

b. `\! pmdmgr -Secss\;T0xffffffff`

- For OVO

  a. `ecsmgr -i 1 -trace 65536`

3. Pump the events through ECS by whatever mechanism
4. The trace file is created in
   - For NNM - `$OV_log/pmd.log0` on HP-UX, Solaris and Linux and `%OV_CONTRIB%\pmd.log0`

NOTE Turning tracing ON will have an impact of performance and the trace file size - so use it only when you need to debug.

Turn Off Tracing

To turn off tracing, type

```
ecsmgr -fact_update Composer \
/opt/OV/contrib/ecs/CO/CompTraceOff.fs
```

Reading the Trace Message

After pumping in the event(s) grep for the string "`Composer`" to get the trace messages. There are trace messages printed when the event enters each of the phases mentioned above:

1. The first phase is Alarm Signature processing - The start of processing is demarcated by the string "`Alarm Signature processing starting`" and the end of this processing is demarcated by "`Alarm signature processing done`". The lines between these two contain information related to Alarm Signature processing.
2. The second phase is the Advanced Filter processing (assuming that the filter criteria of atleast one Correlator was met). This section of processing is demarcated by the strings, "`Advanced signature processing starting`" and "`Advanced signature processing done`".
3. Assuming the alarm passes the Advanced Filter then the individual logic for the Correlators are invoked. Look for the strings "`Input processing starting`" and "`Input processing done`". Between these two points the input processing of the individual Correlators is

executed. Look for the strings "`Executing logic for the Correlator - starting`" and "`Executing logic for the Correlator - done`". The 5th field, "`Correlator Name`", will have the name of the Correlator that is being executed. There may be a number of such pairs depending on the number of Correlators to be executed for the alarm.

4. Look for the string "`Decision after all the Correlators run`" to find out the fate of the event after all the Correlators have processed it.
5. If any of the Correlators decide to HOLD the event, the processing of the event continues after the specified period. (In the meanwhile other events may enter the system - check the EventID field to ensure that you are looking at the trace messages for the right event). Look for the strings "`HOLD processing starting`" and "`HOLD processing done`" to find the demarcations for this phase. (Note: There may multiple such phases depending on how many Correlators requested for holding the event.)
6. Look for the string "`Decision after all the Correlators run`" to find out the fate of the event after all the Correlators have processed it.
7. The variables defined in a Correlator are periodically printed out as the Correlator is being processed. Look for the string "`The variables are`" - below this line is the set of variables that were defined for the Correlator and the value currently bound to it. If the variable is yet to be evaluated it will show "`Not Yet Evaluated`". Variables starting with "__" maybe ignored as they are used internally.
8. To find invocation of synchronous functions look for "`Calling`". The line will contain the function being called and the arguments being passed to it.
9. Look for the string "`Return value of the function`" to find out what the synchronous function returned.
10. For asynchronous functions look for "`Asynchronously invoking function`" to find out the function being invoked and the parameters being passed to it. For return values from asynchronous functions look for "`Setting variable`". The string will indicate the variable being set and for which Correlator. (Asynchronous functions are handled by common code so field 5 in the trace message will be set to "`Common`".

11. To trace the flow of an event, type,

    `grep “Composer”  | grep <`*`eventid`*`>`

    where,

    `eventid` is the “if” of the event that needs to be traced.

12. To trace the actions of a specific Correlator, type,

    `grep “Composer” |grep <`*`correlatorname`*`>`

    where,

    `correlatorname` is the name of the Correlator that needs to be traced.

13. To trace the actions of a Correlator on a given `event id, type,`

    `grep “Composer” |grep <`*`correlatorname`*`> | grep <`*`eventid`*`>`

Checklist of faults

Before attempting to diagnose a problem, consider whether one of the following common situations may be the cause.

- **Error messages**

  The Composer displays an error-trap when the Composer encounters problems during processing. The sample error-trap sent by the Composer is as shown below:

```
Trap-PDU {
  enterprise {1 2 3 4},
  agent-addr internet : “\x7F\x00\x00\x01”,
  generic-trap 6,
  specific-trap 5000,
  time-stamp 0,
  varaible-bindings {
    {
      name{1 2 3 4 5},
     value simple: string : “Correlator Name is unctionnotpresent : Event Input -
Asynchronous function :’C’ function not found. Path = libEcho - function is
HelloWorld”
   }
 }
}
Trap-PDU {
  enterprise {1 2 3 4},
  agent-addr internet : “\x7F\x00\x00\x01”,
  generic-tra 6,
  specific-trap 5000,
  time-stamp 0,
  varaible-bindings {
    {
      name{1 2 3 4 5},
      value simple: string : “Asynchronous function call/s had errors or times
out : Correlators affected are[annoNodeTimeOutInput]”
   }
 }

}
```

The general form of the error string displayed is of the form

`Correlator Name is` *`Correlatorname`*`:`*`where`*`:`*`reason`*

where:

`Correlatorname` is the name of the Correlator that has caused the error

`where` indicates at what point in the processing the Correlator encountered the error and could be one of the following

- Post 'HOLD' - The logic of a Correlator may choose to HOLD an event for a period of time. For example, a User-Defined Input function may choose to HOLD the alarm for 10 seconds. After 10 seconds the event exits from the HOLD and the Post HOLD processing begins. In this example, it would involve invoking the Output function.
- Asynchronous User Defined Output
- Asynchronous variables used in alter/create
- Processing parameters for asynchronous functions
- Event Input - Asynchronous function
- Event Input
- Processing Advanced Filter
- Event Cleanup

`reason` is the cause of the problem. The section below describes some of the commonly displayed error messages.

Some of the conventions used in the sample error messages are explained below:

Convention	Description
CORRELATORNAME	The name of the Correlator that has caused the problem
FUNCTIONNAME	The name of the function that is called
LIBRARYNAME	The name of the library that contains the user-supplied function
ARGUMENTS	The parameters passed to the function
VARIABLENAME	The name of the variable being evaluated

Commonly displayed error messages

Message

`Function returned an Error - Error returned is Path =` *`LIBRARYNAME`* `- Function is` *`FUNCTIONAME`*`: Function returned:` *`ERROR STRING RETURNED BY THE FUNCTION`*

Cause

The called external function returned an error. The function was invoked as part of a evaluating variable in the Correlator *CORRELATORNAME*.

Message

`'c' Function not found: Path =` *`LIBRARYNAME`* `- Function is` *`FUNCTIONAME`*

Cause

The C function could not be found in the library specified. Either the library name or the function name specified is incorrect.

Message

`Arguments to function is invalid Path =` *`LIBRARYNAME`* `- Function is` *`FUNCTIONAME`*`: Function returned:` *`PARAMETERS TO THE FUNCTION`*

Cause

The datatypes of the parameters passed to the function are invalid. The valid data types that can be passed to a function are, integer, real, string, Object ID, and time.

Troubleshooting the Composer during Runtime

Message	`Library not found Path = `*`LIBRARYNAME`*` - Function is `*`FUNCTIONAME`*
Cause	The library path is not found. You could have entered an incorrect library name or the path specified is incorrect. Check the logs to see what the problem is.
Message	`Function timed out Path = `*`LIBRARYNAME`*` - Function is `*`FUNCTIONAME`*
Cause	The called function returned but has not yet invoked the callback function to indicate completion of processing.
Message	`Function timed outPath = `*`LIBRARYNAME`*` - Function is `*`FUNCTIONAME`*
Cause	The called function returned but has not yet invoked the callback function to indicate completion of processing.
Message	`Function returned an Error - Error returned is Path = `*`LIBRARYNAME`*` - Function is `*`FUNCTIONAME`*`: Function returned: `*`ERROR STRING RETURNED BY THE FUNCTION`*
Cause	The user supplied function returned an error while processing.
Message	`Unable to evaluate variable - `*`VARIABLENAME`*`: No definition seems to exist for variable `*`VARIABLENAME`*
Cause	Incorrect usage of automatic variable is most likely the cause. For example using OutputRetval as a parameter to the input function in User-Defined correlation would result in this error.

Message	`Lookup entry not found for key KEYNAME: Unable to evaluate VARIABLENAME`
Cause	The key for which a datastore lookup is being performed cannot be found and hence the variable cannot be evaluated.
Message	`Asynchronous function call/s had errors or timed out: Correlator is -` *`CORRELATORNAME`*`: variables are -` *`VARIBALENAMES`*`: Correlator is -` *`CORRELATORNAME`*`: Variables are -`*`VARIABLENAME`*
Cause	The called function(s) has not returned values in the expected time. The most likely cause is that the function invoked did not send a return value.
Message	`Type mismatch while creating/altering the event. Attribute, Value pair is: (`*`ATTRIBUTENAME, VALUE BEING ASSIGNED`*`)`
Cause	The variable type being assigned to the attribute is incorrect. For example assigning a string to the specific-trap. Refer to Table C-1 on page 238.
Message	`Variable Binding value not specified for index` *`INDEXNUM`*
Cause	The value for the variable bindings with index number *`INDEX NUM`* has not been provided.
Message	`Variable Binding Name not specified for index` *`INDEXNUM`*
Cause	The name for the variable bindings with index number *`INDEXNUM`* has not been provided.

10 Correlation Composer for NNM

This chapter contains information you require to efficiently use the HP OpenView Correlation Composer in the NNM environment.

Introduction

In the NNM environment, Correlation Composer is available through the ECS Configuration Window.

NNM provides a set of built-in Correlator Stores to enable maintenance of Correlators specific to that environment. For information on these Correlators refer to *HP OpenView Network Node Manager's Managing Your Network with HP OpenView Network Node Manager*. These Correlators are loaded and enabled when NNM is installed and can be enabled or disabled at any time through the Correlation Composer window.

Composer in the Operator Mode

To start the Correlation Composer in the NNM environment, do the following.

1. In the Event Configuration window, select `Edit->Event Correlation`. The web browser displaying the Correlations is displayed.

2. Select the row with Composer and select the `[Modify]` button. The Composer window is displayed. The Correlation Composer window opens.

When the Composer is launched from NNM, permissions to certain menus, Correlators, and Correlator fields are restricted. In particular, operators will only be able to:

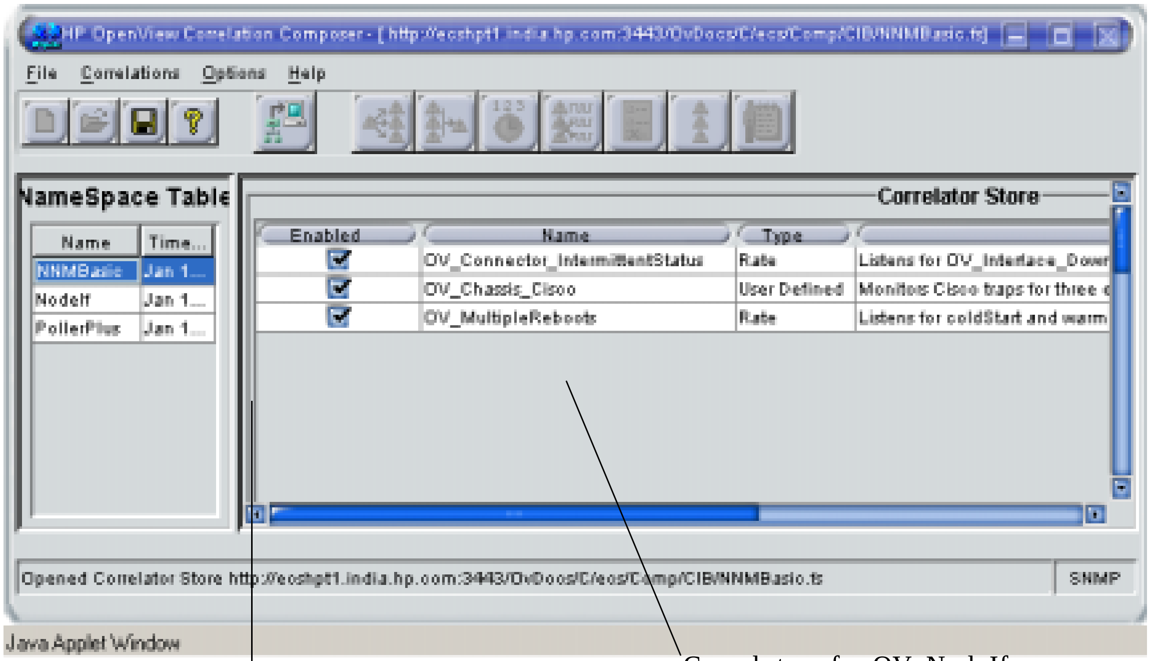
- View those Correlators defined for the listed NameSpaces. By default, only the NNM built-in correlators are visible in the Composer GUI.

- Edit those parameters defined as editable. By default, almost all of the correlator parameters for the NNM built-in correlators are not editable.

- Save changes made to the Correlators by clicking the Save shortcut menu or selecting `File->Save`. Saving a Correlator Store file saves it to the `$OV_CONF/ecs/CIB` directory. You must deploy the Correlator Store to apply its changes to the ECS engine.

- Deploy Correlator Stores to the ECS engine by clicking the `Deploy` shortcut menu or selecting `Correlations->Deploy`.
- Launch the Online help contents by clicking the `Help` shortcut menu or selecting `Help->Table of Contents`.

Figure 10-1 Composer in the Operator's mode


HP OpenView Correlation Composer - [http://ecshpt1.india.hp.com:3443/OvDocs/C/ecs/Comp/CIB/NNMBasic.fs]
File
Correlations
Options
Help
NameSpace Table
Correlator Store
Java Applet Window
SNMP
NNM built-in Correlator Stores
Correlators for OV_NodeIf

Name	Time...
NNMBasic	Jan 1...
NodeIf	Jan 1...
PollerPlus	Jan 1...

Enabled	Name	Type	
✓	OV_Connector_IntermittentStatus	Rate	Listens for OV_Interface_Down
✓	OV_Chassis_Cisco	User Defined	Monitors Cisco traps for three c
✓	OV_MultipleReboots	Rate	Listens for coldStart and warm

Opened Correlator Store http://ecshpt1.india.hp.com:3443/OvDocs/C/ecs/Comp/CIB/NNMBasic.fs

The Composer in the NNM environment restricts the usage of some of the menu options. The table summarizes the options available.

Table 10-1 Menu Items available for Composer in NNM Environment

Menu Item	Description
`File->Save`	Saves the Correlator Store. By default NNM ships four Correlator Stores: `NNMBasic.fs`, `NodeIf.fs`, `PollerIntermittent.fs` and `PollerLinkDown.fs`
`File->Close`	Closes the Correlator Store.
`File->Exit`	The Composer window is closed.

Table 10-1 Menu Items available for Composer in NNM Environment

Menu Item	Description
`Correlations->Global Constants`	Opens the Global Constants window and allows to edit the values of Global Constants that are declared editable in the Security file. For more information on the Security file refer to “Security File” on page 124.
`Correlations->Deploy`	Deploys the Correlator Stores listed in the NameSpace file. By default the NameSpace file lists the Correlator Stores `OV_NNM_BASIC` and `OV_NodeIf`. The Deploy procedure involves merging the Correlator Stores into a single Correlator Store and load this file into the ECS engine. For more details on The Deploy procedure refers to the Deploy Configuration file. for more details on how to create the Deploy Configuration file refer to “Step 4: Creating the Deploy Configuration file” on page 129.
`Options-> Forcefully Unlock`	Provides mutually exclusive access to the Correlator Store. For more details refer to “Mutual Exclusive Access to Correlator Store files” on page 141.
`Options->Appearance`	Displays a submenu for selecting the kind of Look and Feel of the interface.
`Options->View/Restore Backup`	Displays a submenu to select the version of backed up file. For more details refer to “View Backup Files” on page 67. In the NNM Environment, Backup files have read and write permission for all.
`Help->Overview`	Displays the Composer Online Help.
`Help->Table Of Contents`	Displays the Composer Online Help Table of Contents. You can also view the Help Index from this window.
`Help->About Correlation Composer`	Displays the current release and copyright information for the Composer and associated software.

Composer in the Developer Mode

The Composer in the Developer mode in the NNM environment operates the same way it operates in the Standalone mode. To start the Composer in the Developer mode, type

`ovcomposer -m d`

The Developer besides creating correlation logic also maintains configuration files required by the Composer. The definitions provided in these files govern how Composer functions in the NNM environment.

What are these files?

Default Configuration files required by the Composer are shipped along with the product. The files for usage in the NNM environment are:

- **NameSpace Configuration file**

  The default NameSpace file resides in the directory `$OV_CONF/ecs/CIB`. This file does not contain any entries and must be edited by the NNM user. The listing in this file governs what will be displayed in the Composer window.

Example 10-1 Enable the Operator to view the Correlator Stores

By default, when NNM is installed, a NameSpace file is placed on the NNM system that provides access to the built-in NNM Correlator Stores.

The NameSpace file resembles the text below:

```
#Comments begin with '#'
#Configure this file as per your requirements
#The format of the namespace.conf file is as follows:

#<logical name>=<associated file>
#where,
#<logical name> is the logical name as displayed in the namespace table when
#Correlation Composer operates in Operator Mode.
#<associated file >is the file associated with the logical name. All the files
#are relative to $OV_CONF/ecs/CIB/ directory.
#Example
#OV_Basic=OV_Basic/OV_Basic.fs

OV_NNM_Basic=NNMBasic.fs
```

```
OV_NodeIf=NodeIf.fs
```

- **Security file**

  A default security file is created for every Correlator Store when it is saved. This file resides in the same directory where the Correlator Store is saved. Typically, `$OV_CONF/ecs/CIB` and/or subdirectories below it.

Example 10-2 Enable the Operator to edit the parameters `Window Period` and the `Count` of alarms

If you want to provide the flexibility to the Operator to be able to edit the values of Window Period and Count for the Correlator `OV_Connector_IntermittenStatus`, edit the Security file `NNMBasic.sec`.

The NNMBasic's Security file now resembles the text below:

```
#NOTE: No space between the comma seperating the variable fields
#ALL_TEMPLATE=ALL_PARAMS

OV_Connector_IntermittentStatus=WINDOW,COUNT
```

- **Deploy Configuration file**

  The default Deploy Configuration file resides in the directory `$OV_CONF/ecs/CIB`. The entries in this file are the default entries required by the Composer in the NNM environment.

Built-In Function

The following build-in function is provided to work only with SNMP traps and can be used only in the NNM environment.

getOIDValue

Syntax

`getOIDValue` *`oid failValue`*

Where:

`oid` is the name of variable-binding for which the value is to be extracted

`failValue` is the value that is returned by the function if the retrieve fails.

Description

The `getOIDValue` function retrieves the value of the first occurance of the corresponding name from the variable-bindings.

Examples

Consider the Trap as

```
Trap-PDU
enterprise {1 2 3 4 995},
agent-addr internet : “\x0A\x00\x01\x7F”
generic-trap 6,
specific-trap 95,
time-stamp 414746291,

variable-bindings{

    {

      name { 1 3 6 1 4 1 11 2 7 2 17 0},
      value simple : number : 95

    },

    {
      name {1 3 6 1 4 1 11 2 17 2 2 0},
      value simple : number : 96
    },

     {
      name {1 3 6 1 4 11 2 17 2 17 0},
      value simple : number 97
     }

   }
}
```

The statement such as

```
getODValue 1.3.6.1.4.11.2.7.2.17.0 -1
```

returns the value 95 if the retrieve is successful, otherwise the function returns -1. Note that the OID and failvalue must be declared as variables in the Correlator definition section. Refer to “Function” on page 83 for details on how to use Functions.

A Ready Reckoner

This chapter briefly provides information on the various sections to be configured for the different Correlator Templates.

Note that this is only a quick reference and for any additional information required, refer to the corresponding chapters. This chapter discusses the following:

- “Parameters for Correlator Templates” on page 217
- “Flow of Events” on page 225
- “Built-In functions” on page 226
- “Terminology Flow” on page 224

Parameters for Correlator Templates

Enhance Correlator Template

Enhance Correlation defines parameters to add more information to an alarm before it is output. The table below summarizes the functionality of all the parameters in the Enhance Correlator Template window.

Table A-1 **Enhance Correlator Template**

Button Name	Selection Mandatory?	Selected	Functionality
`Want Original`	No	Yes	The original alarm is output along with the enhanced alarm
		No	Only the enhanced alarm is output, the original alarm is discarded.
`Enhance Always`	No	Yes	Alarms will be modified and output regardless of any other correlation deciding to discard this alarm
		No	Alarms will be modified if and only if no other Correlator decides to discard this alarm.

Multi-Source Correlator Template

The Multi-Source Correlator Template is used to correlate events of different kinds and output events with enhanced information. The table below summarizes the functionality of all the parameters in the Multi-Source Correlator Template window.

Table A-2 Multi-Source Correlator Template

Button Name	Selection Mandatory?	Selected	Functionality
`Discard on Set Completion`	No	Yes	The alarm will be discarded if the set is complete, else will be forwarded
		No	The alarm will be forwarded regardless of set completion.
`Window Period`	Yes	Yes	The time period within which all alarms of the set need to arrive for the set to be considered complete. The alarms can arrive in any order.
`Set`	Yes	No	Operates in Mode 1. Refer to “Multi-Source Correlator Template” on page 25
		Yes	Operates in Mode 2. Refer to “Multi-Source Correlator Template” on page 25

Rate Correlator Template

The Rate Correlator Template maintains a count of event arrival and outputs a new event based on this count. The table below summarizes the functionality of all the parameters in the Rate Correlator Template window.

Table A-3 Rate Correlator Template

Button Name	Selection Mandatory?	Selected	Functionality
`Window Period`	Yes		Time period for which the alarm arrival rate is monitored.
`Count`	Yes		The threshold count. If the number of alarms exceeds threshold count within the specified Window Period then the rate threshold is considered breached.
`Discard`	No	Yes	All alarms are discarded. Only the new alarm, if created, is output.
		No	Alarms are not discarded. Additionally the new alarm, if created, is output.

Repeated Correlator Template

The Repeated Correlator Template window is used to define parameters to discard events of similar type. The table below summarizes the functionality of all the parameters in the Repeated Correlator Template window.

Table A-4 Repeated Correlator Template

Button Name	Selection Mandatory?	Selected	Functionality
`Window Period`	Yes		Time period for which the alarm duplication is monitored
`Discard Duplicate`	No	Yes	Chooses Mode 1 of operation. Refer to "Repeated Correlator Template" on page 27
		No	Chooses Mode 2 of operation. Refer to "Repeated Correlator Template" on page 27
`Discard Immediately`	No	Yes	This is applicable only if the Discard Duplicate button is chosen. The effect of this is that all duplicate alarms will be discarded without participating in other correlations.
		No	Duplicate alarms will be discarded only after participating in other correlators.

Suppress Correlation Template

The Suppress Correlator Template allows a specific category of alarms to be discarded.

A key point to be noted is that the New Alarm Creation tab in the Correlator is disabled as there are no new alarms to be created.

The table below summarizes the functionality of all the parameters in the Suppress Correlator Template window

Table A-5 Suppress Correlator Template

Button Name	Selection Mandatory?	Selected	Functionality
`Participate in Other Correlation`	No	Yes	The alarm does not participate in other correlations before it is discarded.
		No	The alarm takes part in other correlations, before it is discarded.

Transient Correlation

Transient Correlation correlates events based on some threshold values.

A key point to be noted is New Alarms can be defined only after a threshold value has been defined.

The table below summarizes the functionality of all the parameters in the Transient Correlator Template window.

Table A-6 Transient Correlator Template

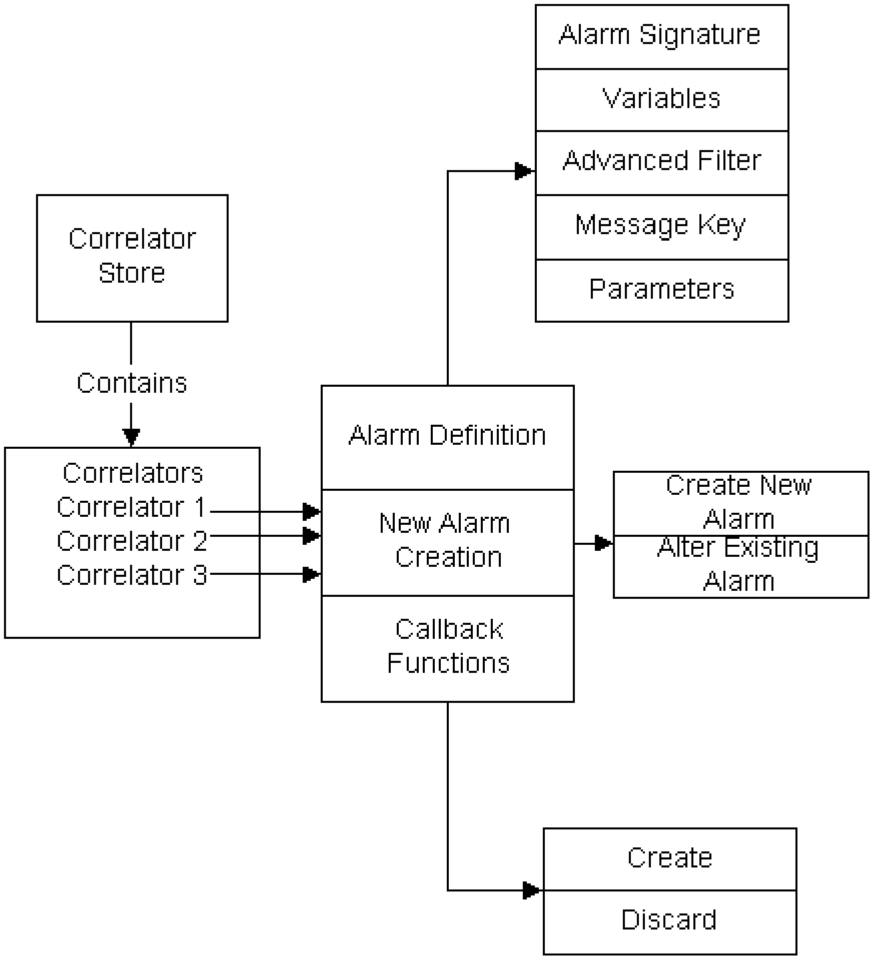
Button Name	Selection Mandatory?	Selected	Functionality
`Window Period`	Yes		The maximum time a Failure alarm is held by the Composer waiting for a Clear Alarm. If the Clear alarm is received while the alarm is held, both the Clear and Failure alarms are discarded. If no Clear alarm is received in this Window Period, the failure alarm is forwarded. Typical hold periods are between 1-10 minutes depending on the severity of the problem.
`Enable Threshold`	No	Yes	Maintains a count of the number of alarm pairs for the specified threshold Window. If the count equals the Threshold Count within the Threshold Window, a new alarm is created and forwarded.
		No	No Count is maintained. The Threshold Count and Threshold Windows are both disabled.

Table A-6 Transient Correlator Template (Continued)

Button Name	Selection Mandatory?	Selected	Functionality
Threshold Count	Yes, if Enable Threshold has been selected		This button is enabled only when the Enable Threshold button has been enabled. The number of alarm pairs, viz the Failure and Clear alarms arriving
Threshold Window	Yes, if Enable Threshold has been selected		This button is enabled only when the Enable Threshold button has been enabled. The time period for which the count is maintained

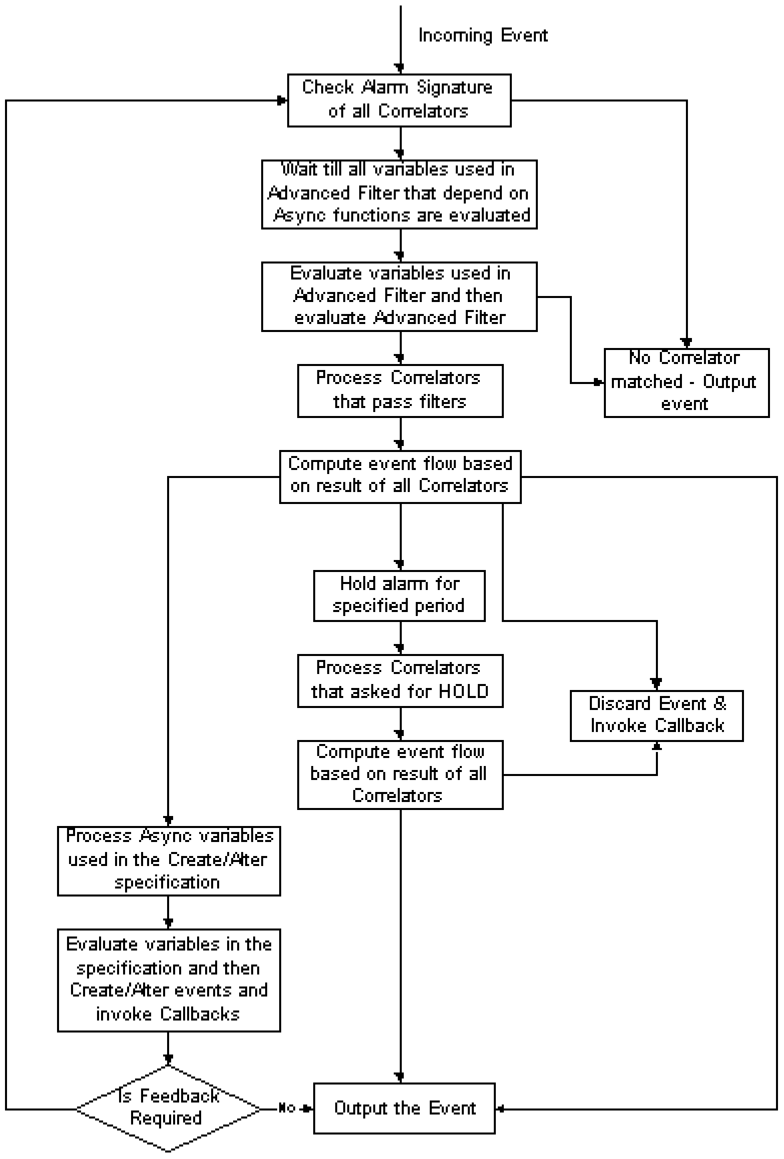
Terminology Flow

The figure below depicts the different sections of a Correlator.



Flow of Events

The figure below summarizes the Flow of Events.


Incoming Event
Check Alarm Signature of all Correlators
Wait till all variables used in Advanced Filter that depend on Async functions are evaluated
Evaluate variables used in Advanced Filter and then evaluate Advanced Filter
No Correlator matched - Output event
Process Correlators that pass filters
Compute event flow based on result of all Correlators
Hold alarm for specified period
Process Correlators that asked for HOLD
Discard Event & Invoke Callback
Compute event flow based on result of all Correlators
Process Async variables used in the Create/Alter specification
Evaluate variables in the specification and then Create/Alter events and invoke Callbacks
Is Feedback Required
No
Output the Event

Built-In functions

add

Syntax

add *int1 int2*

Where:

int1 and *int2* are integers

bitand

Syntax

bitand *int1 int2*

Where:

int1 and *int2* are integers

bitinv

Syntax

bitinv *int*

Where:

int is an integer

bitor

Syntax

bitor *int1 int2*

Where:

int1 and *int2* are integers

bitxor

Syntax

bitxor *int1 int2*

Where:

int1 and *int2* are integers

div

Syntax *int1* div *int2*

Where:

int1 is the integer dividend.

int2 is the integer divisor.

getByIndex

Syntax getByIndex *list index failvalue*

Where:

list is a list of any data types

index is the position from which the value is to be extracted

failvalue is the value extracted if the getByIndex function fails

getCounter

Syntax getCounter *toInit key1, key2 ...*

Where,

toInit is the method in which the value will be retrieved

key1, key2,... are the keys based on which the value is retrieved

The field *toInit* can take the following values

STORE_INIT	The stored value is returned and the storage memory occupied by this value is freed.
STORE_NO_INIT	The stored value is returned, but the storage space is not deleted and further calls to retrieve will return the stored value.

getHour

Syntax `getHour()`

Description The `getHour` function returns the current hour. The result is an integer and value can be between 0-23. All time is represented in UTC.

getMinute

Syntax `getMinute()`

Description The `getMinute` function returns the current minute. The result is an integer and value can be between 0-59. All time is represented in UTC.

getMonth

Syntax `getMonth()`

Description The `getMonth` function returns the current month. The result is an integer and value can be between 1-12.

getTime

Syntax *`getTime ()`*

Description The `getTime` function returns the time in seconds since epoch(1 January 1970). The result is a string.

makeList

Syntax makeList *`arguments`*

Where:

`arguments` is the list of arguments that are passed to the function

mod

Syntax *`int1`* `mod` *`int2`*

Where:

int1 and *int2* are both integers.

mul

Syntax

`mul` *int1 int2*

Where:

int1 and *int2* are two integers

retrieve

Syntax

`retrieve` *toInit failvalue key1, key2, ...*

Where:

toInit is the method in which the value will be retreived

failvalue is the value that is returned by the function if the retrieve function fails.

key1, key2, ... are the keys based on which the value is retrieved

retrieveStr

Syntax

`retrieveStr` *toInit failvalue key1, key2,...*

Where:

toInit is the method in which the value will be retrieved

failvalue is the value to be returned by the function if the retrieve function fails.

key1, key2,... are the keys based on which the value is retrieved

setCounter

Syntax

`setCounter` *toInit increment window key1, key2,...*

Where:

toInit is the method in which the value will be set.

increment is the increment value.

window is the time period for which the value will be stored.

key1, key2,... are the keys against which the values will be set.

store

Syntax store *value window key1, key2, ...*

Where:

value is the value that is to be stored

window is the time period for which the value will be stored

key1, key2, ... are the keys based on which the value is stored

storeStr

Syntax storeStr *toAppend seperator value window key1, key2, ...*

Where,

toAppend parameter decides how the value will be stored

seperator is the field seperator

value is the value to be stored

window is the time period for which the value will be stored

key1, key2 ... are the keys based on which the value will be stored

sub

Syntax sub *int1 int2*

Where :

int1 and *int2* are the any two integers

B Error Messages

This chapter provides a list of messages you may encounter while using the Correlation Composer and the corrective action, if any, that you should take to overcome the error.

Error Messages displayed while creating Correlator Store files

Message: `No blank entry allowed`

Cause	You have left an entry blank
Action	Enter valid values in the blank cell.

Message: `Invalid Syntax`

Cause	You have entered some invalid characters like consecutive blanks, commas or mismatched quotes.
Action	Enter values without blank spaces, commas and matched quotes.

Message: `Variable in use, cannot delete`

Cause	You have tried to delete a variable which is used as a input to define other variables.
Action	Change the condition rule for the variable and delete the variable.

Message: `Variable Name cannot be Null`

Cause	You have not entered the name for the Variable
Action	Enter the name for the variable

Message: `Duplicate Variable Name`

Cause	Specified Variable Name already exists.
Action	Provide a new name for the variable

Message: `Variable Name in use, cannot rename`

Cause	You have tried to rename the variable which i used as input while defining other variables
Action	Change the condition rule for the variable and rename the variable.

Message: `Unspecified Correlator Name`

Cause	You have not entered a name for the Correlator.
Action	Enter a name for the Correlator.

Message: `Unspecified function name`

Cause	You have not entered the name for the function while defining Callback functions
Action	Enter the name of the function

Message: `Correlator name is invalid`

Cause	You have entered a blank space or special characters in the Correlator name text field.
Action	Rename the Correlator name without any blank spaces or special characters

Message: `Unspecified WindowPeriod`

Cause	The time period for which the correlation has to be maintained has not been entered.
Action	Enter the tim period for which the correlation has to be entered in the Time Period window.

Message: `WindowPeriod should be integer only`

Cause	You have tried to enter special characters or alphabets in the Time Period window.
Action	Renter the Time Period in numeric form only.

Message: `Unspecified ThresholdCount`

Cause	In Transient correlation, you have tried to correlate without specifying the count of the Failure and Clear pairs after checking the Enable Threshold checkbox.
Action	Enter the Threshold Count

Message: `Unspecified ThresholdWindow`

Cause	In Transient correlation, you have tried to correlate without specifying the time period for which you want to monitor the correlation.
Action	Enter the time period for which you want to monitor the correlation.

Message: `Duplicate Alarm Name`

Cause	Specified Correlator name already exists.
Action	Enter a different name for the Correlator.

`Alarm Name is use, cannot delete`

Cause	In Multi-Source correlation, you are trying to delete an existing alarm whose attributes are being used to create a new alarm.
Action	Change the New Alarm Specification and delete the alarm.

`Message: Minutes in WindowPeriod cannot be greater than 60.`

Cause	You have tried to enter the minute value greater than 60.
Action	Covert minutes to hours and enter the value in the Window Period field.

`Message: Seconds in WindowPeriod cannot be greater than 60.`

Cause	You have entered the second value greater than 60.
Action	Convert second value to minute and enter the value in the Window Period field.

`Message: Threshold Count should be integer only`

Cause	You have tried to enter special characters or alphabets in the Threshold Count window.
Action	Renter the Threshold Count in numeric form only.

`Message: Look and Feel not supported`

Cause	You have tried to change the `Look and Feel` to `“Windows”` on an Unix machine.
Action	Renter the Threshold Count in numeric form only.

`Message: Unknown Event Type`

Cause	The Event type specified while starting the Orchestrator from the command line is invalid.
Action	Enter the valid Event Types namely, SNMP, CMIP, OpC or X733.

Error Messages displayed while Deploying Correlator Store files

The following errors are displayed when you try to deploy the Correlator Store into the ECS Engine.

`Message: Merge failed: Cannot execute the csmerge script.`

Cause The `csmerge` script (that is invoked internally at the time of deploy) could not be executed.

Action

`Message: Cannot load the Correlator Store into the ECS engine.`

Cause The Correlator Store cannot be loaded into the ECS engine.

Action Check if the ECS engine is running. If it is already running, contact the Administrator for more details.

`Message: Merge Failed. Correlator Stores are of different Event Type.`

Cause The `csmerge` file is trying to merge Correlator Store files created for different Event types.

Action Such files cannot be merged.

`Message: Merge Failed. Correlator Stores have different Perl files.`

Cause The Perl files are different in the Correlator Stores.

Action Include all Perl files into one main Perl file and reference this while specifying the Perl filename.

`Message: Merge Failed. Correlator Stores have different C Libraries.`

Cause The C libraries are different in the Correlator Stores

Action Reference only one C library while specifying C library name.

`Message: Merge Failed. Cannot open file.`

Cause	One of the Correlator Store files specified in the Namespace files cannot be opened.
Action	Check the file permissions. Change file permissions for it to be visible appropriatly.

`Message: Merge Failed. Destination File already exists.`

Cause	A Correlator Store file with the same name already exists.
Action	Provide a different name to the merged Correlator Store.

C Event Attributes

This appendix contains the valid Event attributes for all the Event Types supported by the Composer.

Table C-1 Event Attributes

Message Attribute	Type	Description
CMIP		
managedObjectClass	String	This is the registered class of the managed object that emitted the event.
managedObjectInstance	String	This identifies the managed object instance that emitted the event.
eventTime	Time	This is the time at which the event was created by the managedobject emitting it.
eventType	Integer	This is the registered NOTIFICATION for the event.
eventInfo	String	This is the generic place-holder for details of the specific event, as defined by the eventType.
eventInfo.probableCause	Integer	This is the probable cause of the alarm.
eventInfo.perceivedSeverity	Integer	This is the perceived severity of the alarm.
eventInfo.additionalText	String	This is the additional text that may be extracted from the message received or added as extra information to the alarm.
OpC		
AACTION_ACK	Boolean	Defines whether or not the message is acknowledged automatically on the OVO management server after the corresponding automatic action has finished successfully.
AACTION_ANNOTATE	Boolean	Defines whether or not OVO creates “start” and “end” annotations for automatic actions.

Table C-1 Event Attributes (Continued)

Message Attribute	Type	Description
AACTION_CALL	String	The command to use as automatic action for the OVO message. Default: empty string; max. length: 2000 chars.
AACTION_NODE	String	Defines the system on which the automatic action runs. Default value: the content 'NODENAME'; max. length: 254 chars
AACTION_STATUS	Integer	Defines the status of the automatic action belonging to the current message. Possible values are: • 0 - ACTION_UNDEF (default) • 1- ACTION_DEF (default if AACTION_CALL is defined) • 2 - ACTION_STARTED • 3 - ACTION_FINISHED
APPLICATION	String	Application name to use for the OVO message. Default: empty string; max. length: 32 chars.
CREATION_TIME	Time	The time the message was created. The time is in UNIX format (seconds since Epoch). Default: the (local) time when the message was created.
FORWADED		Read only. Defines whether or not message is forwarded in an environment configured with manager-to-manager forwarding.
GROUP	String	The OVO message group to use for the message. Default: empty string; max. length: 32 chars.

Table C-1 Event Attributes (Continued)

Message Attribute	Type	Description
INSTR_IF	String	The name of the external, instruction-text interface. The external, instruction-text interface must be configured in OVO. Default: empty string; max. length: 36 chars.
INSTR_IF_TYPE	Integer	Defines whether the internal OVO instruction-text interface or an external interface is used todisplay instructions for the message. Possible values are: • 0 - INSTR_NOT_SET(default) • 1 - INSTR_FROM_OPC(instruction stored in OVO database) • 2 - INSTR_FROM_OTHER (instruction accesses via external instruction-text interface)
INSTR_PAR	String	Parameters for the call to the external, instruction-text interface. Default: empty string; max. length: 254 chars.
MSGID	String	Read only. Unique identifier of the message. Modified or newly created messages will assume the ID: ‘00000....’
MSGSRC	String	Read only. This attribute specifies the source of the message, example, the name of the encapsulated logfile if the message originated from logfile encapsulation or the interface name if the message was sent via an instance of the Message-Stream Interface. Default: empty string; no max. length.

Table C-1 Event Attributes (Continued)

Message Attribute	Type	Description
MSGSRC_TYPE	Integer	Read only. Specifies the source type of the message. Each source is represented in one bit, for example, a message that was generated by the logfile encapsulator and then modified at the Agent MSI will have 'bit' or LOGFILE_SRC and AGTMSI_SRC set. Possible values are: • 1 - CONSOLE_SRC (MPE/iX source) • 2 - OPCMSG_SRC • 4 - LOGFILE_SRC • 8 - MONITOR_SRC • 16 - SNMPTRAP_SRC • 32 - SVMSI_SRC (MSI on OVO management server) • 64 - AGTMSI_SRC (MSI on OVO managed node) • 128 - LEGLINK_SRC (Legacy Link interface) • 256 - SCHEDULE_SRC
MSGTEXT	String	Message text. Default: empty string; no max. length.
MSGTYPE	String	This attribute is used to group messages into subgroups, example, to denote the occurrence of a specific problem. Default: empty string; max. length: 36 ASCII chars, no spaces.

Table C-1 Event Attributes (Continued)

Message Attribute	Type	Description
MSG_LOG_ONLY	Boolean	Inserts the message immediately into the history-message log when the message is received on the Management Server. The message is not sent to any operator. An Operator will only be able to see the message when using the OVO history message browser
MSI_OUTPUT	Integer	Defines the handling of the message in the OVO Message Stream Interface. Each value is representing one bit that can be bitor'ed. Possible values are: • 0 - SV_MSI_NO_OUTPUT (default) • 1 - SV_MSI_DEVERT • 2 - SV_MSI_COPY • 0 - AGT_MSI_NO_OUTPUT (default) • 4 - AGT_MSI_DIVERT • 8 - AGT_MSI_COPY
NODENAME	String	The name of the system on which the message was created. The message is only handled by the OVO management server if NODENAME is part of the OVO managed environment (OVO node bank). Default: local node name; max. length: 254 chars.
NOTIFICATION	Boolean	Forwards OVO messages from the OVO management server to the OVO Trouble-ticket Notification Service interface(s), if the appropriate notification interface(s) is/are configured and active.

Table C-1 Event Attributes (Continued)

Message Attribute	Type	Description
OBJECT	String	The “object” name to use for the OVO message. Default: empty string; max. length: 254 chars.
OPACTION_ACK	Boolean	Defines whether the message is acknowledged automatically on the OVO management server after the corresponding operator-initiated action has finished successfully.
OPACTION_ANNOTATE	Boolean	Defines whether or not OVO creates “start” and “end” annotations for the operator-initiated action.
OPACTION_CALL	String	Command to use as operator-initiated action for the OVO message. Default: empty string; max. length: 2000 chars.
OPACTION_NODE	String	Defines the system on which the operator-initiated action should run. Default value: NODENAME; max. length: 254 chars.
OPACTION_STATUS	Integer	Defines the status of the operator-initiated action belonging to the current message. Possible values are: • 0 - ACTION_UNDEF (default) • 1 - ACTION_DEF (default if OPACTION_CALL is defined) • 2 - ACTION_STARTED • 3 - ACTION_FINISHED • 4 - ACTION_FAILED

Table C-1 Event Attributes (Continued)

Message Attribute	Type	Description
ORIGMSGTEXT	String	The original message text. This attribute allows you to set additional source information for a message. It is useful if the message text was reformatted but the OVO operator needs to have access to the original text. Default: empty string; no max. length.
READ_ONLY		Read only. Defines whether or not a message is forwarded as a "notification" in an environment configured with manager-to-manager forwarding.
RECEIVE_TIME	Time	Read only. Defines the time the message was received by the management server. The time is in UNIX format (seconds since Epoch). This value is set by the management server.
SERVICE_TIME	Time	This is the Service time of the message.
SEVERITY	Integer	The severity of the message. Possible values are: • 4 - SEV_UNKNOWN • 8 - SEV_NORMAL • 16 - SEV_WARNING • 64 - SEV_MINOR • 128 - SEV_MAJOR • 32 - SEV_CRITICAL
TIME_ZONE_DIFF	Time	Read only. Defines the difference in seconds between UTC and local time at the time a message is created.

Table C-1 Event Attributes (Continued)

Message Attribute	Type	Description
TROUBLETICKET	Boolean	Defines the forwarding of OVO messages from the OVO management server to the OVO trouble-ticket (TT) interface, if the TT interface is configured.
TROUBLETICKET_ACK	Boolean	Defines that the OVO management server acknowledges the message automatically if forwarding of the message to the trouble ticket system was successful.
UNMATCHED	Boolean	Defines whether or not the message matched a condition.
SNMP		
enterprise	Object ID	Identifies the network management subsystem that generated the trap.
generic-trap	Integer	One of the predefined values in the definition. Values must be between 1 and 6
specific-trap	Integer	A code that indicates the nature of the trap more specifically than the generic-trap number. Specific trap numbers are defiend by the owning enterprise and are meanigful only in conjunction with the enterprise attribute.
time-stamp	Integer	The number of time ticks, in hundreths of a second, between the last intialization of the device and the generation of the trap. Genrally meaniglesss for correlation purposes.

Table C-1 Event Attributes (Continued)

Message Attribute	Type	Description
variable-bindings	String, Integer	Additional information about the trap. The contents of this field is dependent on the enterprise ID and specific-trap values.
agent_addr	String in dot notation	The network address of the object that generated the trap in the form of an ECDL tuple.
X733		
ServicePrimitive	Integer	Defines the service primitive of the current event message.
Format	Integer	Defines the format of the current event message.
ProcessName	Integer	
InvokedIdentifier	Integer	
FmpType	Integer	
Mode	Integer	Defines the mode of the event message.
ManagedObjectClass	String	This is the field that indicates the managed object class of the object emitting the alarm.
ManagedObjectInstance	String	This field indicates the instance of the MOC that emitted the alarm.
EventType	Integer	
EventTime	Time	This field indicates the date and time at which the alarm was emitted. Mapping information to this field is not mandatory.

Table C-1 Event Attributes (Continued)

Message Attribute	Type	Description
ProbableCause	Integer	This is the probable cause of the alarm. X.721 and M.3100 are supported for probable cause global variables. The probable causes for each of these formats are listed in Appendix C, “List of Probable Causes,” on page 463 of this manual.
PerceivedSeverity	Integer	This is the perceived severity of the alarm.
BackedUpStatus	Integer	Defines the back up status of the ebeent message. The valid values are: • NotBackedUp • BackedUp
TrendIndication	Integer	Defines the trend of the alarm severity change. The valid values are: • LessSevere • NoChange • MoreSevere
SpecificProblems	String	This is the specific problem that caused the alarm. This corresponds to the specific problems defined in the managed object class definition.
NotificationIndentifier	Integer	This is a notification id generated by the device that emitted the alarm.
AdditionalText	String	This is the additional text that may be extracted from the message received or added as extra information to the alarm being transferred to the FM Server.
AdditionalInformation	String	This is also additional information that can be sent with the alarm to the FM Server.

Table C-1 Event Attributes (Continued)

Message Attribute	Type	Description
ERId	Integer	Internal use. Do NOT modify.
AlarmId	String	Internal use. Do NOT modify.
CorResult	Integer	Internal use. Do NOT modify.
AdminState	Integer	Internal use. Do NOT modify.
TransientCount	Integer	Internal use. Do NOT modify.
RelatedCount	Integer	Internal use. Do NOT modify.
PortName	String	This is the name of the port at which the alarm was received.
NetworkEquipShortname	String	This is the short name of the parent network element of the object that emitted the alarm.

D Pattern Matching

This chapter explains the Pattern Matching conditions provided by the Composer.

Pattern-Matching

ECS provides a powerful text pattern-matching language that allows logical testing for the existence of substrings and patterns. Parts of a text string can be extracted and assigned to tags, which may be reused within the same scope. This section describes the operators and syntax of the pattern-matching language.

The pattern-matching language used in the match functions is the same as that used in HP OpenView IT Operations.

Frequently, pattern-matching means simply scanning for a specific substring in the target string. For example, to search for the substring ERROR anywhere in the target string you search for the pattern:

```
"ERROR"
```

Similarly, should you wish to match text not containing a specific substring (for example, WARNING), you type:

```
"<![WARNING]>"
```

This uses the not operator "!", together with the chevrons "< >" that must enclose all operators, and the square brackets "[ ]" that isolate sub-patterns.

You control case-sensitivity with a separate argument to the `Match.make` function.

Defining Match Expressions

- **Ordinary Characters**

  Ordinary characters generally represent themselves. However, if any of the following special characters are used they must be prefaced with a backslash escape character (\ ) to mask their usual function.

[ ] < > | ^ $

- **Expression Anchoring Characters (^ and $)**

  If the caret (^) is used as the first character of the pattern, only expressions discovered at the beginning of lines are matched. For example, "^ab" matches the string "ab" in the line "abcde", but not in the line "xabcde".

  If the dollar sign is used as the last character of a pattern, only expressions at the end of lines are matched. For example, "de$" matches "de" in the line "abcde", but not in the string "abcdex".

  If ^ and $ are not used as anchoring characters, that is, not as first or last characters, they are considered as ordinary characters without masking.

- **Expressions Matching Multiple Characters**

  Patterns used to match strings consisting of an arbitrary number of characters require one or more of the following expressions:

  - <*> matches any string of zero or more characters (including separators)

<*n**> matches a string of *n* arbitrary characters (including separators)

  - <#> matches a sequence of one or more digits
  - <*n*#> matches a number composed of *n* digits
  - <S> matches a sequence of one or more separator characters
  - <*n*S> matches a string of *n* separators
  - <@> matches any string that contains no separator characters, in other words, a sequence of one or more non-separators; this can be used for matching words.

Separator characters are configurable for each pattern. By default, separators are the space and the tab characters. The separator string is specified as the second element in the 3-tuple passed to the `Match.make` function.

- **Bracket ([and]) Expressions**

  The brackets ([and]) are used as delimiters to group expressions. To increase performance, brackets should be avoided wherever they are superfluous. In the pattern:

  ```
  "ab[cd[ef]gh]"
  ```

  all brackets are unnecessary—`"abcdefgh"` is equivalent.

  Bracketed expressions are used frequently with the OR operator "`|`", the NOT operator "`!`" and when using sub-patterns to assign strings to tags.

- **The OR (|) Operator**

  Two expressions separated by the vertical bar character "`|`" matches a string that is matched by either expression. For example,

  ```
  "[ab|c]d"
  ```

  This matches the string `"abd"` and the string `"cd"`.

- **The NOT (!) Operator**

  The not operator "`!`" must be used with delimiting square brackets, for example:

  ```
  "<![WARNING]>"
  ```

  The pattern above matches all text which does not contain the string `"WARNING"`.

  The not operator may also be used with complex sub-patterns:

  ```
  “LN<*>: R< ![490|[501[a|b]]] >-<*>”
  ```

  The above pattern makes it possible to generate a message for any line connection other than from repeaters 490, 501a or 501b. Therefore, the following would be matched:

  ```
  "LN270: R300-427"
  ```

  However, this string is not matched, because it refers to repeater 501a:

  ```
  "LN270: R501a-800"
  ```

If the sub-pattern including the not operator does not find a match, the not operator behaves like a <*>: it matches zero or more arbitrary characters. For this reason, there is a difference between the UNIX expression `"[!123]"`, and the corresponding ECS pattern matching expression: `"<![1|2|3]>"`. The ECS expression matches any character or any number of characters, except 1, 2, or 3; the UNIX expression matches any *one* character, except 1, 2, or 3.

- **The Mask (\ ) Operator**

  The backslash “\” is used to mask the special meaning of the characters:

  `[ ] < > | ^ $`

  A special character preceded by \ results in an expression that matches the special character itself.

  Because ^ and $ only have special meaning when placed at the beginning and end of a pattern respectively, you need not mask them when they are used within the pattern (in other words, not at beginning or end).

  The only exception to this rule is the tab character, which is specified by entering “`\t`” into the pattern string.

Tags

Search patterns may use tags to identify part(s) of the target string. For example, to compose a new string from selected parts of the target string. define a tag, add “.*tagname*” before the closing chevron. The pattern:

```
^errno: <#.number> - <*.error_text>
```

matches a string such as:

```
errno: 125 - device not in service
```

and assigns `"125"` to the tag `number` and `"device not in service"` to the tag `error_text`. The tags may be accessed as members of a dictionary.

Assignment Rules

In matching the pattern "<*.tag1><*.tag2>" against the string "abcdef", it is not immediately clear which substring of the input string is assigned to each tag. For example, it is possible to assign an empty string to tag1 and the whole input string to tag2, as well as assigning "a" to tag1 and "bcdef" to tag2, and so forth.

The pattern-matching algorithm always scans both the input line and the pattern definition (including alternative expressions) from left to right. <*> expressions are assigned as few characters as possible. <#>, <@>, <S> expressions are assigned as many characters as possible. Therefore, tag1 will be assigned an empty string in the above example.

To match an input string such as:

```
"this is error 100: big problem"
```

use a pattern such as:

```
error <#.errnumber>:<*.errtext>
```

In which:

- "100" is assigned to the tag errnumber
- "big problem" is assigned to the tag errtext

For performance and pattern readability purposes, you can specify a delimiting substring between two expressions. In the above example, “:” is used to delimit <#> and <*>.

Matching <@.word><#.num> against "abc123" assigns "abc12" to word and "3" to num, as digits are permitted for both <#> and <@>, and the left expression takes as many characters as possible.

Patterns without expression anchoring can match any substring within the input line. Therefore, patterns such as:

```
"this is number<#.num>"
```

are treated in the same way as:

```
"<*>this is number<#.num><*>"
```

Sub-Patterns Assignment

In addition to being able to use a single operator, such as * or #, to assign a string to a tag, you can also build up a complex sub-pattern composed of a number of operators, according to the following pattern: `<[`*`sub-pattern`*`].tag>`

For instance: `<[rack<#>.brd<#>].hware>`

In the example above, the period (`.`) between `rack<#>` and `brd<#>` matches a similar dot character, while the dot between `]` and `hware` is necessary syntax. This pattern would match a string such as `"rack123.brd47"` and assign the complete string to `hware`.

Other examples of sub-patterns are:

```
<[Error|Warning].sev>
```

and

```
<[Error[<#.n><*.msg>]].complete>
```

In the first example above, any line with either the word “Error” or the word “Warning” is assigned to the tag, `sev`. In the second example, any line containing the word “Error” has the error number assigned to the tag, `n`, and any further text assigned to `msg`. Finally, both number and text are assigned to `complete`.

Examples of Pattern-matching Conditions

The following examples show some of the many ways in which the ECS pattern-matching language can be used.

- `"Error"`

  Recognizes any message containing the keyword `Error` at any place in the message, when ExactCase is specified.

- `"panic"`

  Matches all messages containing `panic`, `Panic`, `PANIC` at any place in the text, when IgnoreCase is specified.

- `"logon|logoff"`

  Uses the or operator to recognize any message containing the keyword `logon` or `logoff`.

- `"^switch:<*.msg> errno<*><#.errnum>$"`

  Recognizes any message such as:

```
"switch: lost service errno : 6"
```

or

```
"switch: service unavailable; errno 16"
```

In the first example, the string `"lost service errno"` is assigned to the tag `msg`. The digit `6` is assigned to the tag `errnum`. Note the way that the anchoring symbol is used to specify that the digit `6` will only be matched if it is at the end of the line.

- `"^errno[ |=]<#.errnum> <*.errtext>"`

  Matches strings such as:

```
"errno 6 - no such device or address "
```

or

```
"errno=12 not enough capacity. "
```

Note the space before the or operator. The expression in square brackets matches either this blank space, or the equals sign. The space between `<#.errnum>` and `<*.errtext>` is used as a delimiter. Although not strictly required for assignments to the tags shown here, this space serves to increase performance.

- `"^system:<*>:<*.id>:"`

Matches a line delimited by colons such as:

```
"system:abc123:#103.79a:270295114730:"
```

and returns the third field in tag `id`. The colon “`:`” in the middle of the pattern is used to delimit the string passed to `id` from the preceding string. The colon at the end of the pattern delimits the string passed to `id` from the succeeding field in the input pattern. Here the colon is necessary to separate the strings, not merely to enhance efficiency.

- `^Warning:<*.text>on circuit<@.circuit>$`

Matches any message such as:

```
"Warning: too many errors on circuit 473-186"
```

and assigns `"too many errors"` to `text`, and `"473-186"` to `circuit`.

Glossary

Advanced Filter Alarms that have entered a rule, can be further filtered based on the Advanced Filter Condition. This condition is typically used to define filters based on external factors like state and topology.

Alarm Signature The Alarm Signature forms the first level of filtering based on event attributes. Further processing takes place when an event matches all attributes set in the Alarm Signature. The Alarm Signature is a set of data structures consisting of `Attribute` name, `Operator` and `Value`

Attributes An alarm is a set of name value pairs where the name is referred to as an attribute.

Callback Functions A correlation rule applied to an event can result in the event getting discarded or a new correlation being output. User-defined functions can be invoked to perform user specific functions like logging of events. Callback functions can be invoked at the time of creating a new alarm or when the alarm is discarded.

Combine Variables are combined to form a new variable by using the Combine operator.

Constant Constant values are used for reference while defining a correlation rule. The variable name is bound to the value specified in the value field.

Correlation Correlation is the processing of an event stream to improve its value, perhaps by making it smaller and perhaps by improving its information content. This processing is performed on the basis of relationships between events.

Correlation Composer The HP OpenView Correlation Composer is a combination of a packaged ECS circuit and the graphic user interface used to parametrize and define Correlation Rules to perform Event Correlation.

Correlator A Correlator uniquely identifies a unit of correlation logic to be applied to an event or a set of events.

Correlator Store A Correlator Store contains a set of Correlators which define correlation requirement.

Enhance Enhance Correlation enables event attributes of an event to be added, deleted or changed after correlation, resulting in a change of information content of an event.

Event Correlation Services ECS can identify and highlight changes in the state of a network (by suppressing and correlating event storms), and then pass on or generate events which have more significance or a higher value.

Extract Extract - Sub strings within the event attribute can be extracted.

Function The return value of an user defined function can be bound to a variable name. The variable type can be a function whose value will be associated to the name of the variable.

Global Constants Values can be bound to names and referred to by this name while defining Correlation Rules.

Lookup Values from datastore can be quarried and bound to variables using the Lookup operator. The return type of the Lookup operator is always the same type as the value in the Datastore. However, if more than one parameter is specified, the return value will be a combined string.

Message Key The message key identifies the instance of the rule under which the alarm is correlated after the alarm has passed both the Alarm Signature and Advanced Filter conditions.

Multi-Source Multi-Source correlation is used to define a relationship between an arbitrary number of alarms, potentially from different sources, that together form a logical set that identifies the problem. The set is considered complete if all alarms configured arrive within the specified time window.

Parameters Parameters are set to change the default behavior of the basic rule type. The functionality of the parameters can vary across the different Correlation types.

Rate Rate correlation allows you to measure the number of events occurring in a defined window of time (called the threshold time) and outputs an event with more information content. The rate is maintained for a particular category of events. However, if the arrival rate exceed the threshold, it is indicative of a serious problem and a new alarm is created and forwarded.

Repeated Operators will not find it necessary to receive alarms of the same kind during a defined window of time. Similar alarms can be eliminated and only an alarm containing useful information can be passed on.

Suppress Suppress Correlation is used when a specific category of alarms has to be discarded. The alarm signature specified identifies these alarms.

Transient Alarms coming from the same network element with the same probable cause and specific problem within a specified time period(window) are considered transient. A period of time can be defined for which this transience must occur i.e. you are checking for the number of transient alarms occurring in a threshold time. If the transient suppression exceeds the threshold then a new alarm will be generated and forwarded to the operator notifying him of threshold breach and the number of transients suppressed in that window

Variables Variables are names given to values to be used while defining the Correlation Rule.