

# HP Correlation Composer Software

for the HP-UX, Linux, Solaris, and Windows operating systems

Software Version: 9.00 and higher

---

## User's Guide for HP Operations Manager and HP Network Node Manager

Document Release Date: May 2014

Software Release Date: September 2010



## Legal Notices

### Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

### Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

### Copyright Notices

© Copyright 2004–2014 Hewlett-Packard Development Company, L.P.

### Trademark Notices

Adobe® and Acrobat® are trademarks of Adobe Systems Incorporated.

HP-UX Release 10.20 and later and HP-UX Release 11.00 and later (in both 32 and 64-bit configurations) on all HP 9000 computers are Open Group UNIX 95 branded products.

Intel®, Itanium®, and Pentium® are trademarks of Intel Corporation in the U.S. and other countries.

Java is a registered trademark of Oracle and/or its affiliates.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

Oracle is a registered trademark of Oracle and/or its affiliates.

UNIX® is a registered trademark of The Open Group.

## Audience

This guide explains how to efficiently use HP Correlation Composer with HP Operations Manager (HPOM) and HP Network Node Manager (NNM). It is intended for operations personnel who maintain event correlation in HPOM and NNM environments. These users should have a general operational understanding of managed entities (networks and distributed applications). In particular, they should understand the event types generated by managed entities. Users should also have a good understanding of HPOM or NNM.

## Documentation Updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates, or to verify that you are using the most recent edition of a document, go to the following location:

**<http://h20230.www2.hp.com/selfsolve/manuals>**

Under “Product,” select one of the following:

- **Network Node Manager**
- **Operations for UNIX**
- **Operations for Windows**

This site requires that you register for an HP Passport and log on.

To register for an HP Passport ID, go to the following location:

**<http://h20229.www2.hp.com/passport-registration.html>**

Or click the **New Users – Please Register** link on the HP Passport log-on page.

If you subscribe to the appropriate product support service, you also receive updated or new editions. For details, contact your HP sales representative.

## Related Documents

For more information about Composer, see the following documents:

- **Composer Online Help**

Composer includes an online help system that explains its functionality.

To access the online help system, go to the main window of Composer and click **Help→Overview**.

- **Deploying HP OMi in an HP BSM Solution**

This white paper explains how you can deploy HP Operations Manager i (HP OMi) as part of a larger HP Business Service Management (BSM) solution. In particular, it describes how to customize HP Network Node Manager i (NNMi) messages on an HP Operations Manager (HPOM) for UNIX server to fit HP OMi Topology Based Event Correlation (TBEC).

To find out how to access this white paper from the HP Software Product Manuals website, see [Documentation Updates](#) on page 4.

## Support

Visit the HP Software Support website at the following location:

**<http://www.hp.com/go/hpsoftwaresupport>**

This website provides contact information and details about the products, services, and support that HP Software offers.

HP Software online support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business.

As a valued support customer, you can benefit by using the support website to do the following:

- Search for knowledge documents of interest.
- Submit and track support cases and enhancement requests.
- Download software patches.
- Manage support contracts.
- Look up HP support contacts.
- Review information about available services.
- Enter into discussions with other software customers.
- Research and register for software training.

Most of the support areas require that you register as an HP Passport user and log on. Many also require a support contract.

To register for an HP Passport ID, go to the following location:

**<http://h20229.www2.hp.com/passport-registration.html>**

To find more information about access levels, go to the following location:

**[http://h20230.www2.hp.com/new\\_access\\_levels.jsp](http://h20230.www2.hp.com/new_access_levels.jsp)**

# Contents

1	Composer Overview	21
	Composer Concepts	22
	Events	22
	Alarms	22
	Event Types	23
	Event Actions	23
	Event Flow	24
	Attributes	26
	Correlators	26
	Correlator Store	28
	Correlation Flow	28
	ECS Engine	29
	Composer Modes	30
	Developer Mode (HPOM and NNM)	30
	Operator Mode (NNM Only)	30
	Correlator Templates	31
	Enhance Correlator Template	31
	Multi-Source Correlator Template	32
	Example of Mode 1: Networking Device Fails	33
	Example of Mode 2: Server Crashes	34
	Rate Correlator Template	34
	Repeated Correlator Template	35
	Suppress Correlator Template	35
	Transient Correlator Template	35
	User-Defined Correlator Template	36
	Correlator Template Evaluation Precedence	37

<b>2 Composer GUI</b> .....	39
Correlation Composer Window .....	40
Online Help .....	40
Shortcut Menus .....	41
Localized Descriptions .....	41
Correlator Configuration Window .....	42
Description Tab .....	43
Definition Tab .....	43
Alarm Signature .....	43
Alarm Signatures in NNM .....	45
Alarm Signatures in HPOM .....	46
Variables .....	47
Variable Evaluation .....	50
Automatic Variables .....	51
Advanced Filter .....	51
Message Key .....	52
Example 1: Generating New Router Alarms .....	54
Example 2. Monitoring Interface Failure Rates .....	55
Parameters .....	55
New Alarm Tab .....	56
Callbacks Tab .....	58
Composer Menus .....	59
File Menu .....	60
Correlations Menu .....	61
Options Menu .....	62
Help Menu .....	63
Composer Toolbars .....	64
Standard Toolbar .....	64
Correlator Templates Toolbar .....	65
Deploy Button (NNM Only) .....	66



<b>3</b>	<b>Getting Started</b>	<b>67</b>
	Starting Composer	68
	Starting Composer from NNM	68
	Starting Composer from HPOM	68
	Stopping Composer	69
	Configuring the Correlator Store	69
	Defining Event Attributes	70
	Default Attributes	70
	Changing Mandatory Attributes	71
	Adding Attributes	72
	Backing Up Files	73
	Automatic Backups of Correlator Store Files	73
	Example of Backed-Up Files	74
	Overriding the Number of Backed-Up Files	76
	Restoring Backed-Up Files	77
<b>4</b>	<b>Developing Correlators</b>	<b>79</b>
	Developing Correlator Stores	80
	Creating a Correlator Store	80
	Opening a Correlator Store	80
	Modifying a Correlator Store	81
	Migrating a Correlator Store to Composer 3.3	82
	Configuring Correlator Stores	83
	Defining Event Types	84
	Optional: Defining Global Constants	85
	Value Types for Global Constants	85
	Defining a Global Constant	86
	Deleting a Global Constant	86
	Defining Alarm Correlators	87
	Creating a Correlator	87
	Defining a Correlator	87
	Defining Variable Types	90
	Defining Constant Values	90
	Combining Variables	90
	Extracting Value Patterns	92
	Defining Functions	93
	Validating Function Definitions	95

Optional: Defining New Alarms . . . . .	96
Changing Alarm Attributes . . . . .	96
Creating a New Alarm. . . . .	97
New Alarm Definition Table . . . . .	98
Optional: Creating Callback Functions . . . . .	99
Callback Variables . . . . .	100
Callback Functions . . . . .	100
Automatic Variables . . . . .	100
Setting the Perl File Location . . . . .	101
Managing Correlators . . . . .	102
Opening a Correlator . . . . .	102
Modifying a Correlator . . . . .	103
Deleting a Correlator . . . . .	103
Writing C Functions . . . . .	104
Creating a C Function . . . . .	104
Skeleton Code for C Functions . . . . .	105
Signatures for C Functions . . . . .	107
Parts of C Functions . . . . .	107
Passing Arguments . . . . .	107
Processing Arguments . . . . .	108
Returning Values . . . . .	108
Allocating Space for the Return Values . . . . .	109
Wrapping the Return Values . . . . .	109
Marshalling the Return Values . . . . .	110
Running the Callback Function . . . . .	111
Configuring the UserDevelopedFuncDetails.xml File . . . . .	111
Writing Perl Functions . . . . .	113
Creating a Perl Function . . . . .	113
Skeleton Code for Perl Functions . . . . .	113
Support for Multiple Perl Files . . . . .	115
Including Files on UNIX . . . . .	115
Including Files on Windows . . . . .	115

Creating a Main Perl File .....	116
User-Defined Correlation .....	117
Input Functions .....	117
Output Functions .....	120
Writing a User-Defined Function .....	121
Return Values .....	121
Flag Values.....	121
Skeleton Code for User-Defined Functions .....	122
Merging Correlator Store Files .....	124
Merging Correlator Stores Specified in the Namespace File.....	125
Removing User Descriptions from the Correlator Store.....	125
Merging Correlator Stores .....	126
Configuration File.....	127
Example 1: Clashing Global Constants .....	128
Example 2: Configuration File .....	129
<b>5 Composer in NNM .....</b>	<b>131</b>
Correlator Stores .....	131
Operator Mode .....	132
Starting Composer in Operator Mode .....	132
Operator Tasks .....	133
Operator Menu Options .....	134
Developer Mode .....	135
Starting Composer in Developer Mode .....	135
Configuration Files .....	135
Namespace File.....	136
Security File.....	136
Deploy Configuration File .....	137
Built-In Function .....	137
getOIDValue.....	137

<b>6 Composer in HPOM</b> .....	139
Composer GUI .....	140
ECS Engine .....	141
Message Correlation .....	142
Correlation Options .....	142
Correlation Guidelines .....	143
Additional Correlation Tools .....	144
Starting the Composer GUI .....	144
Configuring MSI in HPOM for UNIX or Linux .....	145
Configuring MSI on the HPOM for UNIX or Linux Management Server .....	145
Configuring MSI on HPOM for UNIX or Linux Managed Nodes (Agents) .....	146
Configuring MSI in HPOM 9.00 for Windows .....	149
Configuring MSI on the HPOM 9.00 for Windows Management Server .....	149
Configuring MSI on HPOM 9.00 for Windows Managed Nodes (Agents) .....	151
Merging and Deploying Correlator Store Files .....	154
Location of Correlator Store Files .....	154
Composer Applications on UNIX .....	154
Merging and Deploying on the Management Server .....	155
Merging and Deploying on Managed Nodes (Agents) .....	156
Accessing External Data .....	157
Data Store File .....	157
Location of the Data Store File .....	157
Syntax of the Data Store File .....	158
Example 1: Creating a New Data Store .....	158
Example 2: Updating an Existing Data Store .....	159
Perl Scripts .....	159

7 Use Cases in NNM .....	161
Case 1: Enhance Correlation .....	162
PDU for a Temperature Alarm .....	162
Responding to the Temperature Alarm .....	163
Defining the Enhance Correlator Template .....	164
Case 2: Multi-Source Correlation .....	165
PDU for SS7 Link Failure .....	165
PDU for SS7 Link Set Failure .....	165
Responding to the SNMP Trap PDU Alarms .....	166
Defining the Multi-Source Correlator Template .....	168
Case 3: Rate Correlation .....	170
PDU for Radio Antenna Failure .....	170
Responding to Radio Antenna Failure Alarms .....	171
Defining the Rate Correlator Template .....	172
Case 4: Repeated Correlation .....	175
PDU for Duplicate Alarms .....	175
Responding to Duplicate Alarms .....	176
Defining the Repeated Correlator Template .....	177
Case 5: Suppress Correlation .....	179
PDU for Movement Alarms .....	179
Responding to Movement Alarms .....	180
Defining the Suppress Correlator Template .....	181
Case 6: Transient Correlation .....	182
PDU for PCM Link Failure .....	182
PDU for PCM Clear Alarm .....	182
Responding to PCM Link Failure .....	183
Defining the Transient Correlator Template .....	185
Case 7: Multi-Event Correlation .....	188
PDU for MSC Failure .....	188
PDU for BSC Failure .....	188
Responding to Connected MSC and BSC Failure .....	189
Defining the Multi-Source Correlator Template .....	190

<b>8 Use Cases in HPOM</b> .....	193
Case 1: Enhance Correlation .....	194
Changing Simple Message Text .....	194
Changing the Text of a Simple Message .....	195
Changing Message Text in the Enhance Correlator Template .....	196
Replacing Error and Status Codes with Descriptions .....	200
Replacing an Error or Status Code with a Description .....	201
Replacing a Code with a Description in the Enhance Correlator Template .....	201
Enriching Messages by Using Perl Commands .....	205
Appending Comment Fields to Message Text .....	206
Adding a CMA by Name to an Event .....	207
Increasing Event Severity for Non-Critical Events .....	208
Determining Whether To Suppress Events Based on Maintenance Mode .....	209
Case 2: Suppress Correlation .....	210
Suppressing Message Subsets .....	210
Suppressing the Subset of a Message .....	211
Defining the Suppress Correlator Template .....	212
Case 3: Multi-Source Correlation .....	215
Suppressing Messages on Remote Sites .....	215
Suppressing a Sympathetic Message on a Remote Site .....	216
Suppressing Subsets with the Multi-Source Correlator Template .....	218
Case 4: Rate Correlation .....	225
Detecting DNS Outages .....	225
Responding to a DNS Outage .....	226
Defining the Rate Correlator Template .....	228
Case 5: Transient Correlation .....	233
Generating New Messages .....	233
Smart Message Correlation .....	234
Suppressing High CPU Utilization Messages .....	234
Responding to High CPU Utilization Messages .....	235
Defining the Transient Correlator Template .....	237

9 Developer Mode in NNM .....	241
Administrative Tasks .....	241
Starting Composer in Developer Mode .....	242
Configuring Operator Profiles .....	242
Creating Correlator Stores .....	242
Listing Correlator Stores .....	242
Creating NameSpace and Security Files .....	242
NameSpace Files .....	243
Syntax of the NameSpace File .....	243
Example of a NameSpace File .....	243
Guidelines for NameSpace Files .....	244
Security File .....	244
Syntax of the Security File .....	245
Template Names in CORRELATOR_TEMPLATE .....	246
Token Identifiers in TOK_LIST .....	247
Example of a Security File .....	249
Guidelines for Security Files .....	250
Creating Deploy Configuration Files .....	251
Deploy Procedure .....	251
Example of a Deploy Configuration Files .....	252
Guidelines for Deploy Configuration Files .....	253
Parameters for Deploy Configuration Files .....	254
Defining Operator Access .....	255
Customizing the NameSpace File .....	255
Customizing the Security File .....	256
Customizing the Deploy Configuration File .....	256
Deploying the Correlator Store .....	257
Loading the Correlator Store File to the ECS Engine .....	257
Viewing Errors in the Deploy Status Window .....	257
Deploying Correlator Stores from the Command Prompt .....	258

10 Operator Mode in NNM .....	259
Operator Access Rights .....	259
Starting Composer in Operator Mode.....	260
Locking Files .....	261
File Locking Modes.....	261
File Locking Failure .....	262
Recovering Correlator Stores .....	263
Deploying Correlator Stores .....	264
Loading the Correlator Store File to the ECS Engine .....	264
Viewing Errors in the Deploy Status Window .....	264
Deploying Correlator Stores from the Command Prompt .....	265
A Built-In Functions .....	267
Functions .....	267
add .....	269
bitand.....	269
bitinv .....	270
bitor .....	270
bitxor .....	271
div .....	271
getByIndex .....	272
getCounter.....	273
getHour .....	274
getMinute .....	274
getMonth.....	274
getTime .....	274
makeList .....	275
mod .....	275
mul.....	276
retrieve .....	276
retrieveStr.....	277
setCounter.....	278
store.....	280
storeStr .....	281
sub .....	282
Keys .....	283
Multiple Keys .....	283
Unique Keys .....	283



<b>B Event Attributes</b> .....	285
HPOM Event Attributes .....	286
SNMP Event Attributes .....	293
<b>C Pattern Matching</b> .....	295
Syntax of Pattern Matching .....	296
Expression Delimiter ([ ])	296
Operator Delimiter (< >)	296
OR Operator ( )	296
NOT Operator (!)	297
Mask Operator (\)	298
Matching Expressions .....	299
Matching First or Last Characters .....	299
Matching Multiple Characters .....	300
Matching Tags .....	301
Assigning Substrings to Tags .....	301
Assigning Subpatterns to Tags .....	302
Examples of Pattern Matching .....	303
<b>D Troubleshooting in NNM</b> .....	305
Tracing Events .....	306
Enabling Tracing in Composer .....	306
Enabling Tracing in NNM .....	307
Enabling Tracing for ECS .....	308
Disabling Tracing for ECS .....	308
Tracing the Flow of an Event .....	309
Tracing the Actions of a Specific Correlator .....	309
Tracing the Actions of a Correlator Event ID .....	309
Trace Tools .....	310
Trace Configuration File .....	311
Location of the Trace Configuration File .....	311
Sample Trace Configuration File .....	311
Fields of the Trace Configuration File .....	312
Viewing the Binary Trace Configuration File .....	312

Trace Messages .....	313
Location of the Trace Message File .....	313
Format of the Trace Message File .....	313
Reading the Trace Message File .....	314
Error Messages .....	316
Sample Error Messages .....	316
Syntax for Error Messages .....	317
Conventions for Error Messages .....	318
Sample Error Messages .....	319
<b>E Troubleshooting in HPOM .....</b>	<b>321</b>
Verifying Deployment .....	322
Verifying the ECS Process .....	322
Verifying the MSI Configuration .....	325
Verifying Composer File Deployment .....	326
Message Logging .....	326
Enabling Message Logging in HPOM for UNIX or Linux .....	327
Enabling Message Logging in HPOM 9.00 for Windows .....	328
Sample Message Log File .....	330
Sample Input File .....	330
Sample Output File .....	331
Statistics .....	333
Retrieving Statistics .....	333
Sample Statistics .....	334
Event Tracing .....	335
Enabling Tracing .....	337
Disabling Tracing .....	338
Location of the Trace Message File .....	339
Format of the Trace Message File .....	339
Sample Trace Entries .....	340
Reading the Trace Message File .....	341
Tracing the Flow of a Message .....	343
Tracing the Actions of a Specific Correlator .....	343
Tracing the Actions of a Correlator Event ID .....	344
Error Logging .....	344
Format of the Error Log .....	344
Sample Error Log Entries .....	344

<b>F Error Messages</b> .....	345
<b>Creation Errors</b> .....	346
Alarm Name Is In Use, Cannot Delete .....	347
Correlator Name Is Invalid .....	347
Duplicate Alarm Name .....	347
Duplicate Variable Name .....	347
Invalid Syntax .....	348
Look and Feel Not Supported .....	348
Minutes in Window Period Cannot Be Greater Than 60 .....	348
No Blank Entry Allowed .....	348
Seconds in Window Period Cannot Be Greater Than 60 .....	349
Threshold Count Should Be Integer Only .....	349
Unknown Event Type .....	349
Unspecified Correlator Name .....	349
Unspecified Function Name .....	350
Unspecified Threshold Count .....	350
Unspecified Threshold Window .....	350
Unspecified Window Period .....	350
Variable in Use, Cannot Delete .....	351
Variable Name Cannot Be Null .....	351
Variable Name in Use, Cannot Rename .....	351
Window Period Should Be Integer Only .....	351
<b>Deployment Errors</b> .....	352
Cannot Load the Correlator Store into the ECS Engine .....	352
Merge Failed: Cannot Execute the csmerge Script .....	352
Merge Failed: Cannot Open File .....	353
Merge Failed: Correlator Stores Are of Different Event Type .....	353
Merge Failed: Correlator Stores Have Different C Libraries .....	353
Merge Failed: Correlator Stores Have Different Perl Files .....	353
Merge Failed: Destination File Already Exists .....	354
<b>Glossary</b> .....	355



---

# 1 Composer Overview

HP Correlation Composer is a graphical user interface (GUI) on top of the HP Event Correlation Services (ECS) runtime that enables you to customize predefined correlation logic to meet your requirements in HP Operations Manager (HPOM) and HP Network Node Manager (NNM) environments. To customize predefined correlation logic in Composer, you create correlators. Each correlator represents a unit of logic to be applied to an event or a set of events.

In HPOM and NNM environments, the critical challenge for network and system administrators is to manage the massive amount of information related to network, system, and application problems. This information comes in many forms. It is essential that your management solution help you accurately understand which information is important to present to your operators, which information you can discard, and which information you must provide to specialists, who diagnose very complex problems.

Composer enables you to correlate these network, system, and application problems into specific parameters. The problem-specific correlators provided by Composer uniquely identify units of logic you can apply to events or sets of events. In HPOM, these events are known as “messages.” Composer enables you to tailor the event correlation behavior for correlators that are shipped with HP Software products, fine-tuning them to fit your environment. It also enables you to develop your own custom correlators. You do not need any special programming knowledge to customize correlators in this way.

This chapter describes the following:

- [Composer Concepts](#) on page 22
- [Composer Modes](#) on page 30
- [Correlator Templates](#) on page 31

# Composer Concepts

Before defining correlators in Composer, you must understand the following:

- [Events](#) on page 22
- [Attributes](#) on page 26
- [Correlators](#) on page 26
- [Correlator Store](#) on page 28
- [Correlation Flow](#) on page 28
- [ECS Engine](#) on page 29

## Events

In Composer, an event is an unsolicited notification generated by an agent process in a managed object or by a user action. For example, an agent process can generate an unsolicited Simple Network Management Protocol (SNMP) trap. The term “event” is commonly used in Internet Protocol (IP) environments.

## Alarms

Events received from network elements are known as “alarms.” This term is used commonly in telecommunications environments. The event type specifies the type of alarms that can be handled by Composer.



This document uses the terms “event,” “message,” and “alarm” as follows:

- **“Event”**

Composer is used for “event correlation.” In this document, the term “event” is most commonly used to represent the *generic* concept.

- **“Message” and “alarm”**

The terms “message” and “alarm” are used for event types. In HPOM, event correlation is message correlation. In NNM, event correlation is alarm correlation. However, in both HPOM and NNM, an event can also be a generic term for anything coming from the network.

## Event Types

Composer supports the following event types:

- **Messages**

In HPOM, a message is an event related to a managed object. These events are also known as “OpC messages.” HPOM supports only HPOM messages.

- **SNMP event traps**

NNM supports only SNMP event traps.

Composer supports only one event type for a Correlator Store.

## Event Actions

In Composer, you can output or discard events:

- **Outputting events**

When an event is output, it is visible to end users. For example, it can be listed in an NNM event browser or in an HPOM message browser. Event correlation is often used to suppress unwanted events. However, you can also use it to generate new events when certain conditions are detected. Likewise, you can use correlation to enrich existing events with additional useful information. For example, you can add custom message attributes (CMAs) to an HPOM message.

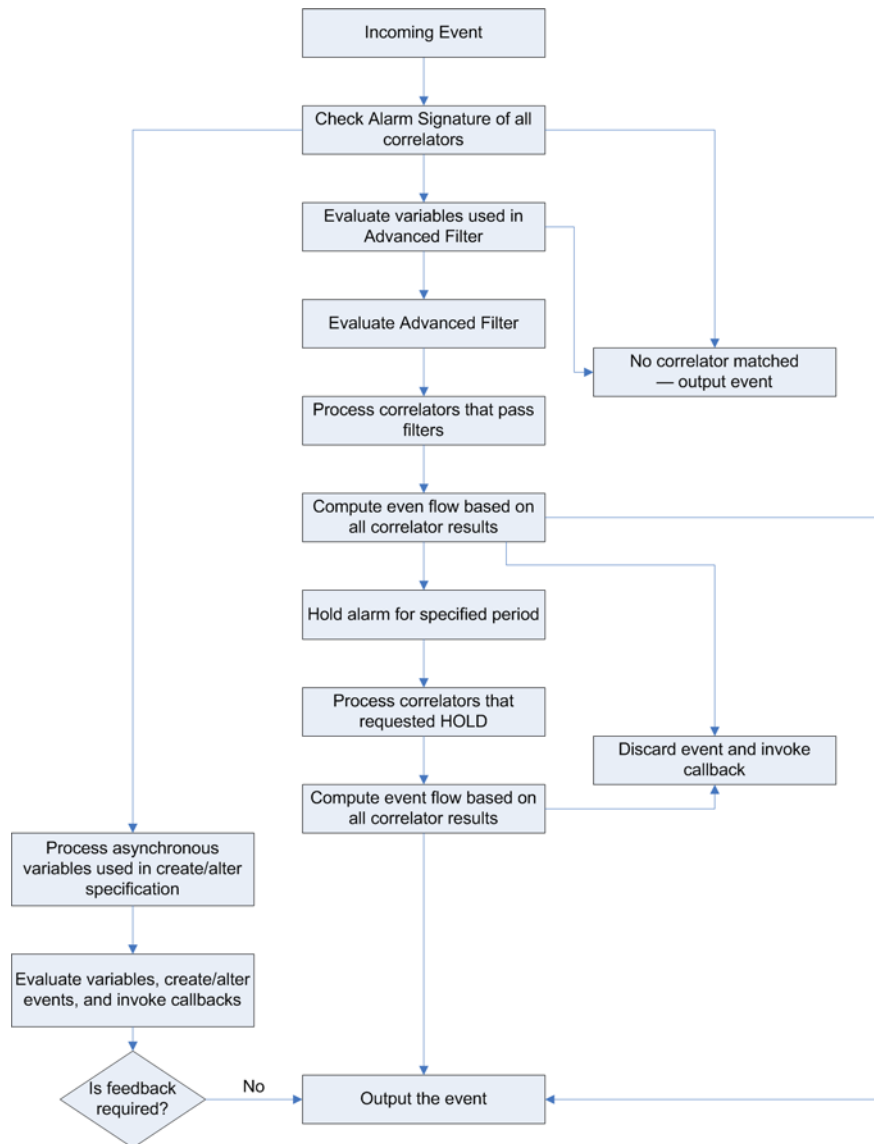
- **Discarding events**

When an event is discarded, it is *not* visible to end users.

## Event Flow

Figure 1 shows how events flow through Composer.

**Figure 1 Event Flow in Composer**





Events flow through the Composer circuit in the following phases:

**1 Comparing with Alarm Signatures**

When an event enters Composer at run time, Composer compares it with the Alarm Signature for all of the correlators. If the event matches none of them, Composer outputs it immediately.

**2 Comparing with Advanced Filters**

Composer compares the event with the Advanced Filters for all of the correlators of the matching Alarm Signatures.

**3 Executing logic**

For each matching filter, the logic of the correlators is executed. Each correlator returns what needs to be done to the event.

**4 Processing actions**

Composer processes the actions of the specified correlators together. If the event is to be held, Composer forwards it to the event hold mechanism.

**5 Holding events**

If Composer holds an event, it outputs the event after the period specified. All of the correlators that requested the event to be held are executed. Each correlator returns what needs to be done to the event.



Events can be held in one of three different ways: `Hold`, `WeakHold`, and `PseudoHold`. For `WeakHold` and `PseudoHold` actions, different correlators may require different hold periods. The output logic of each correlator is executed at the requested times. If multiple correlators require a `Hold` action, the event is held for the longest duration requested.

## Attributes

An event is a set of name value pairs. The name is referred to as an “attribute.”

Attribute examples:

- **NNM**  
enterprise, agent-addr, specific-trap, variable bindings
- **HPOM**  
MSGTEXT, APPLICATION, OBJECT

For a list of attributes for the standard event types, see [Appendix B, Event Attributes](#). You can configure the set of attributes that are visible within Composer. To add CMAs for HPOM, you edit the `CO.conf` configuration file. For details, see [Defining Event Attributes](#) on page 70.

## Correlators

Composer uniquely identifies a unit of correlation logic to be applied to an event or a set of events. This unit of logic is called a “correlator.”

Every correlator has three main sections:

- **Alarm Definition**

In this section, you define the filters, variables, messages keys, and parameters of the correlator.

This section is divided into five subsections:

- ***Alarm Signature***

Primary filter that forms the first level of filtering, based on event attributes. This set of data structures consists of Attribute Name, Operator, and Value. Further processing takes place when an event matches all attributes set in the Alarm Signature.

- ***Variables***

Names assigned to values. After they are assigned, the names can be used in other sections of Composer.

There are two types of variables:

- *Global constants*

Constants defined in the Global Constants section. These constants can be accessed by any correlator within the Correlator Store. For details, see [Correlator Store](#) on page 28.

- *Correlator-specific variables*

Variables defined in the Alarm Definition section. You can access these variables only within the scope of that correlator.

- ***Advanced Filter***

Optional. Secondary filter used to further process alarms that have already been processed by the primary filter, Alarm Signature. Typically, this condition is used to define filters, based on external factors like topology.

- ***Message Key***

Key that identifies the instance of the correlator under which the alarm is correlated. The key 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 that specify the default behavior of the basic correlator template. Typically, these parameter specify the time window for which the correlation is to be monitored.

- **New Alarm**

In this section, you define specifications to create new alarms or alter existing alarms.

- **Callback**

In this section, you configure `create` and `discard` callback functions to establish an audit trail when alarms are created and deleted. For example, when you delete an event, you can invoke a logging function.

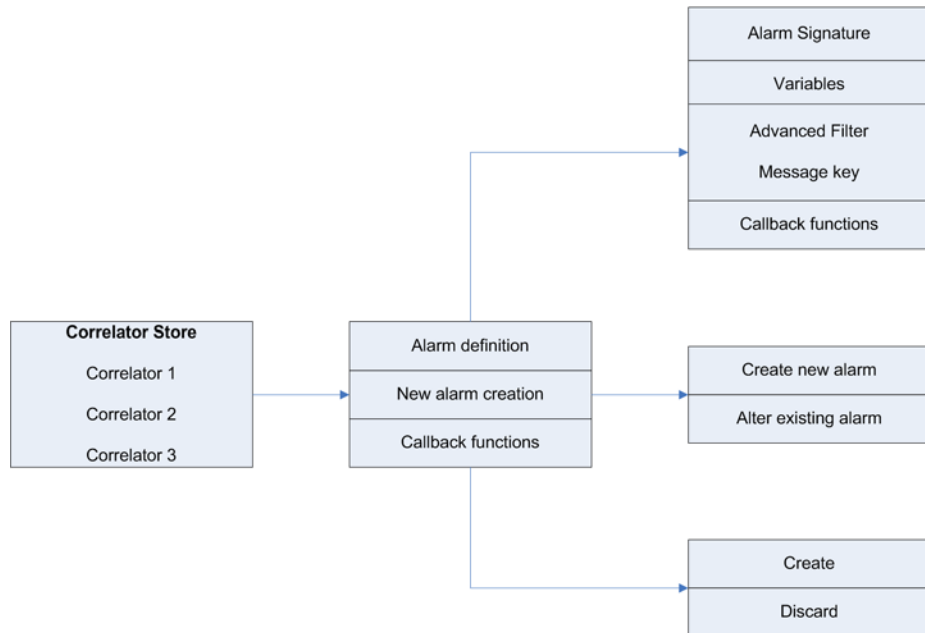
## Correlator Store

A Correlator Store is an ASCII file that stores configured correlators. The Correlator Store is loaded at run time to perform correlation. To find out how to develop a Correlator Store, see [Developing Correlator Stores](#) on page 80.

## Correlation Flow

[Figure 2](#) shows the correlation flow within Composer.

**Figure 2 Correlation Flow**



## ECS Engine

Composer runs inside the HP Event Correlation Services (ECS) engine, which is a component of the following:

- HPOM for UNIX or Linux 9.1x and higher
- HPOM for Windows 9.00 and higher

Composer runs as a single ECS circuit. The ECS engine correlates, suppresses, and enriches HPOM messages, based on the rules configured in deployed circuits, including Composer. For details, see [ECS Engine](#) on page 141.

# Composer Modes

Composer has two modes for developing and maintaining correlators:

- Developer mode (HPOM and NNM)
- Operator mode (NNM only)

## Developer Mode (HPOM and NNM)

In Developer mode, HPOM and NNM developers can set up, create, or modify correlation logic.

Developers can use NameSpace files in both environments:

- **NNM**

NNM developers set up operator access rights by using Security files. Developers determine the area of operation by using NameSpace files.

- **HPOM**

Although HPOM developers can use NameSpace files, these files are designed primarily to provide logic separation within Operator mode, as described in [Operator Mode \(NNM Only\)](#) on page 30. As a result, there is little need for them in HPOM environments. For details, see [Chapter 9, Developer Mode in NNM](#).

## Operator Mode (NNM Only)

In NNM environments, Composer operators determine which correlation logic is active, and set the values of configurable parameters provided by developers. Operators have limited access to Composer functionality. Their access is governed by the permissions specified by developers in the Security files and the area of operation specified in the NameSpace files.

When assigning operator roles, you must set appropriate conditions and permissions to enable operators to efficiently manage the correlation logic deployed. To simplify this task, Composer provides Security and NameSpace files that link operators to parts of the correlation logic.



In Composer, HPOM administrators are treated as operators.

# Correlator Templates

To provide correlation models for the most common correlation tasks, Composer supplies the following correlator templates:

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

In addition, you can use the User-Defined correlator template to configure custom requirements.

## Enhance Correlator Template



You can use the Enhance correlator template to do the following:

- **Create alarms**

Create one or more new alarms. For example, you can create a new alarm that enumerates the set of customers affected by a failed entity.

- **Modify event attributes**

Modify the event attributes of an alarm. For example, you can modify the severity of an alarm, based on the customer who is affected.

- **Add troubleshooting information**

Add information that would be useful to operators for problem resolution or automatic trouble ticketing.

By default, the alarm is enhanced only if no other correlator has discarded the alarm. You can override this default behavior.

When an event is altered, a copy of the original event is made, and then the copy is modified.

Also, the attribute `unique_id` changes:

- *NNM*

UUID attribute (`$u`) of the event changes when altered.

- *HPOM*

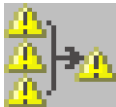
MSGID attribute of the message changes when altered.

When a message is altered (for example, by using the Enhance correlator), and more than one Message Stream Interface (MSI) application is registered, the message is assigned a new Universally Unique Identifier (UUID). However, a new UUID is not desirable in Manager of Manager (MoM) configurations. To ensure that the UUID of a message does not change, you set the configuration variable `OPC_MSI_CREATE_NEW_MSGID` to 4.

For example, for the Server MSI, you would set the configuration variable as follows:

```
# ovconfchg -ovrg server -ns opc -set  
OPC_MSI_CREATE_NEW_MSGID 4
```

## Multi-Source Correlator Template



You can use the Multi-Source correlator template to define a relationship and correlate an arbitrary number of related (or “sympathetic”) alarms, potentially from different sources. Together, these alarms form a logical set that identifies the problem. The set is considered complete if all alarms configured arrive within the specified time window.

On set completion, you can use Multi-Source correlation to do the following:

- **Discard subsets**

Discard a subset of alarms.

- **Modify subsets**

Modify a subset of alarms with attributes defined from any or all of the other alarms in the set.

- **Create alarms**

Create one or more new alarms with values called from attributes or predefined variables from the other alarms in the set.



On completion, the set can operate in one of two modes:

- **Mode 1: Complete set**

Default. When the set is deemed complete, the instance of the set remains in a completed state for the duration of the time window. Typically, you use this mode in situations where all alarms from a source can be discarded if caused by the failure of another entity. The correlator template works in this mode when the Set button is *not* selected.

- **Mode 2: Predefined sets**

When the set is deemed complete, the instance of the set is closed immediately. You use this mode when you expect alarms to arrive in predefined sets, and when you want to ensure that a new instance is created for each set. After a new instance is created, the correlation remains open until at least one event for each source has been received. After the last source has received an event, the set is deemed complete.

## Example of Mode 1: Networking Device Fails

When a common networking device (for example, a switch or router) fails, monitoring software may detect and report a series of `interface_down` alarms, as well as an overall `node_down` alarm. Because the device is down, all interface-related alarms from that device can be discarded safely.

In this example, the set has two alarm types:

- `node_down` alarm
- `interface_down` alarm (marked for discarding)

When a `node_down` alarm and an `interface_down` alarm are received, the set is complete. The `node_down` alarm is shown. The `interface_down` alarm is discarded. Any subsequent `interface_down` alarms are discarded until the time window expires.

## Example of Mode 2: Server Crashes

When a server crashes, NNM detects that it cannot communicate with the server. As a result, NNM forwards a `node_down` alarm to HPOM. HPOM also detects that it cannot communicate with the agent on that server and generates a message. When both of these messages arrive, HPOM generates a report message indicating that a server has gone down.

In this example, the set has two alarm types:

- Node Down message (in HPOM, from NNM)
- OpC Agent Not Responding message

If these alarms are received within a given time window, both can be discarded. After set completion (Server Failure), you must create a new alarm. In addition, if a subsequent alarm belonging to the set arrives within the same time window, it is not deleted until the second set is complete. For example, if a `node_down` alarm arrives immediately after the first set is complete, it is not discarded until another OpC Agent Not Responding message arrives.

## Rate Correlator Template



You can use the Rate correlator template to count the number of events occurring within a specified time window. If the count equals the value specified within the time window, the threshold is considered breached and a new alarm is created.

You can configure the Rate correlator template to do the following:

- **Discard all alarms**  
Discard all alarms (regardless of rate). Emit only the newly created alarm when the threshold is breached.
- **Emit all alarms**  
Emit all alarms as they arrive. Emit the newly created alarm (if any) when the threshold is breached.

## Repeated Correlator Template



The Repeated correlator template can operate in one of two modes:

- **Mode 1: Discard duplicate alarms**

Default. Discards duplicate alarms received within the time window of the first alarm. If you want, you can set up the incoming alarm to participate in other correlations before it is discarded. You can also send an update alarm at the end of the time window. Typically, you send an update alarm 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 selected.

- **Mode 2: Keep duplicate alarms**

Does not discard duplicate alarms. If there is a specification for a new alarm to be created, a new alarm is created for every incoming alarm. Typically, you use this mode to send a new alarm to replace the previously sent one, along with the count of duplicate alarms received so far.

## Suppress Correlator Template



You use the Suppress correlator template when you need to discard a specific category of alarms. Alarms that match all of the conditions in both the Alarm Signatures and Advanced Filter are 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. Typically, you use the Transient correlation template to detect transient failures. When a transient failure is detected, associated events are discarded. You can also use this template to monitor the rate of 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.

For example, `Temperature_ON` and `Temperature_OFF` alarms are generated when the temperature of a router exceeds the threshold or falls below the threshold. You can use the Transient correlation template to discard both alarms if the `Temperature_OFF` alarm is received within five minutes of the `Temperature_ON` alarm.

## User-Defined Correlator Template



You use the User-Defined correlator template to implement a requirement when none of the other correlation models, either by itself or in combination, can meet the correlation requirement.

Alarms that meet the conditions specified in the Alarm Signature and Advanced Filter invoke the input function specified.

The input function can be of the Perl or the built-in type. The return value of the input function determines the action to be taken on the alarm (for example, create a new alarm, discard the alarm, hold the alarm for a specified period, and so on).

If the input function requests that the alarm be held, the output function is invoked after the specified time window. The return value of the output function determines the action to be taken on the alarm.

## 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 in a way that admits the same alarm, both correlators are applied to the alarm.

If one event is managed by multiple correlators, the outcome is determined by the following rules:

- **Evaluation order**

Correlator evaluation follows this order:

- a Suppress correlation
- b Repeated correlation
- c All other correlation (in parallel) except Enhance
- d Enhance correlation

- **Event enhancement**

Enhance correlation is run last.

The event is enhanced only if the following is true:

- No other correlator discards the event.
- Enhance Always is *not* enabled in the Enhance correlator template.

- **Event sharing**

Before the Suppress and Repeated correlators discard an alarm, operators can admit the event into other correlators.

- **Event output**

An event is output only if no other correlator has discarded it.



## 2 Composer GUI

HP Correlation Composer is a graphical user interface (GUI) that enables you to customize predefined correlation logic to meet your requirements in HP Operations Manager (HPOM) and HP Network Node Manager (NNM) environments. To customize predefined correlation logic in Composer, you create correlators. Each correlator represents a unit of logic to be applied to an event or a set of events.

This chapter describes the following:

- [Correlation Composer Window](#) on page 40
- [Correlator Configuration Window](#) on page 42
- [Composer Menus](#) on page 59
- [Composer Toolbars](#) on page 64

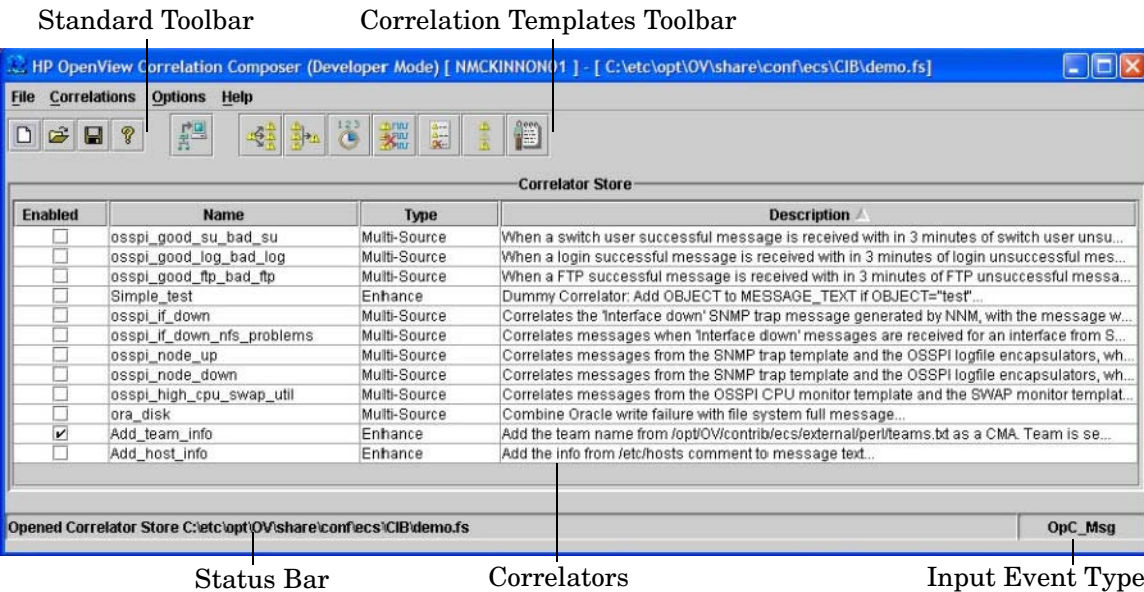


This chapter assumes you are running the Composer GUI in Developer mode.

# Correlation Composer Window

When a Correlator Store opens, the Correlation Composer window displays, as shown in Figure 3. The opening panel of the Correlation Composer window consists of the standard menus and options required to define correlators.

Figure 3 Correlation Composer Window



## Online Help

From the Correlation Composer window, you can access online help as follows:

- To display the online help, press **F1**.
- To display the online help contents, click **Help**→**Table of Contents** from the toolbar.



## Shortcut Menus

From the Correlation Composer window, you can open or learn about menu items with your mouse:

- To select an item, click it.
- To pop up a menu of frequently used options for an item, right-click the item.

## Localized Descriptions

You can open, modify, and use Correlator Stores developed in another locale. User descriptions in the correlators written using the English-language locale are visible in all the other locales, even if the descriptions are not localized in the respective locales. This visibility enables users with knowledge of both the English language and the current locale language to read and localize the English-language descriptions. Because logic is kept separate from language, logic changes are identical across locales. User descriptions change for each locale, as needed.

# Correlator Configuration Window

You define correlators and add their parameters from the Correlator Configuration window, as shown in [Figure 4](#).

**Figure 4 Correlator Configuration Window**

Rate

Name

Description Definition New Alarm CallBacks

**Alarm Signature**

Field	Operator	Value

**Variables**

Name	Type	Value

**Advanced Filter**

Name	Operator	Value

**Message Key**

**Parameters**

**Window Period**

00 hh 00 mm 00 ss Count Discard ☐

OK Cancel Help

The Correlator Configuration window has four tabs:

- [Description Tab](#)
- [Definition Tab](#)
- [New Alarm Tab](#)
- [CallBacks Tab](#)

## Description Tab

In the Correlator Configuration window, the Description tab describes the correlator. The window opens in the Description tab.

## Definition Tab

In the Correlator Configuration window, the Definition tab contains the following sections:

- [Alarm Signature](#)
- [Variables](#)
- [Advanced Filter](#)
- [Message Key](#)
- [Parameters](#)

### Alarm Signature

The Alarm Signature is the primary filter in Composer. As such, it provides the first level of alarm filtering. The Alarm Signature consists of Attribute, Operator, and Value. The operator matches the attribute type to the value entered. Alarms whose attributes match the specification in the Alarm Signature are processed further.

Event attributes vary, depending on the event type selected. For a list of standard event attributes, see [Appendix B, Event Attributes](#). To customize the list of attributes, see [Defining Event Attributes](#) on page 70.

The Alarm Signature supports the following operators:

=

Equal to. Attribute type is equal to the value entered. The value must be an integer, float, string, boolean (true, false), or object ID.

!=

Not equal to. Attribute type is *not* equal to the value entered. The value must be an integer, float, string, boolean (true, false), or object ID.

<

Less than. Attribute type is less than the value entered. The value must be an integer or float.

>

Greater than. Attribute type is greater than the value entered. The value must be an integer or float.

<=

Less than or equal to. Attribute type is less than or equal to the value entered. The value must be an integer or float.

>=

Greater than or equal to. Attribute type is greater than or equal to the value entered. The value must be an integer or float.

matches

Attribute contains the pattern specified by the value entered. The value must be an integer, float, string, or object ID.



The pattern is given as a string. It follows the rules of all patterns used in HPOM.

Example 1:

"1234510" matches 10

The string pattern 10 is found in the string 1234510.

Example 2:

"1234510" matches "^10"

The string pattern "^10" returns False. If the value extracted from the attribute is not a string, the value is converted to a string and the pattern is matched.

### Example 3:

enterprise matches "^1.2.3.4"

If the attribute enterprise contains the 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, enterprise would be 1.2.3.4.

Internally, 1.2.3.4 is converted to the string "1.2.3.4", and this pattern is used to search in the string version of enterprise. However, the pattern would also match an enterprise object ID of 5.6.1.2.3.4, which is not the requirement.

For regular pattern matching expressions, see [Appendix C, Pattern Matching](#).

does NOT match

Attribute does *not* contain the pattern specified by the value entered. The value must be an integer, float, string, or object ID.

### Example:

The value "1020" must *not* be present in the attribute.

is in list

Attribute equals at least one value in the list entered. Values must be listed within brackets ([]).

### Example:

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

is NOT in list

Attribute does *not* equal any value in the list entered. Values must be listed within brackets ([]).

## Alarm Signatures in NNM

In NNM, the Alarm Signature attribute agent-addr in the SNMP trap is represented as a string in the "." notation. For example, the agent-addr attribute is passed to a function as "a.b.c.d", not a.b.c.d.

If you need to set `agent-addr` while creating or altering an alarm, make sure that the variable carrying `agent-addr` has the same format. For example, to set `agent-addr` to the IP address `15.10.76.143`, define a variable whose value is a string such as `"15.10.76.143"`.



The IP address supports IPv6 only.

Variable bindings are a name value pair in which the name is always an object ID. When specifying a new alarm that has variable bindings, you must specify both the name and the value for each variable binding. Variable bindings start with an index of 0.

Alarm Signatures in HPOM

In HPOM, you can use any attributes to define the Alarm Signature. The two most commonly used attributes are Message Type (`MSGTYPE`) and Message Text (`MSGTEXT`). `MSGTYPE` is extremely useful, but only in environments where templates and policies have been configured to use it.

Figure 5 shows an example of an Alarm Signature for messages in which the `MSGTYPE` is `"multi_login_failure"`.

Figure 5 Example of MSGTYPE Alarm Signature

Description	Definition	New Alarm	Callbacks
Alarm Signature			
Field	Operator	Value	
MSGTYPE	=	"multi login failure"	

Figure 6 shows an example of an Alarm Signature that uses a pattern match in which `MSGTEXT` matches `"Bad switch user to <*> by <*>"`.

Figure 6 Example of MSGTEXT Alarm Signature

Description	Definition	New Alarm	Callbacks
Alarm Signature			
Field	Operator	Value	
MSGTEXT	matches	"Bad switch user to <*> by <*>"	

## Variables

Variables are names given to values that are used to define correlators or global constants. All attributes in the alarms can be accessed as variables, in which the variable name is the attribute name.

Correlators use the following variables:

### Constant

Value used as a reference when defining a correlator. The variable name is bound to the value specified in the Value field.

Example:

The variable `ErrStr` is bound to the following value:

```
"Temperature High. Check for A/C Failure."
```

The variable `ErrStr` can be used locally within the correlator under which it is declared.

### Extract

Variable that is extracted from a substring within the event attribute. This variable is then used in the New Alarm and Advanced Filter tabs.

Example:

```
"Bad switch user to <*.to_user> by <*.by_user>"
```

This string matches the following message:

```
"Bad switch user to root by jsmith"
```

The string assigns "root" to a `to_user` variable, and assigns "jsmith" to `by_user` variable.

For more pattern matching examples, see [Appendix C, Pattern Matching](#).

## Function

Data returned from a function. This return value can be bound to a name.

You can call functions synchronously or asynchronously. For details, see [Writing a User-Defined Function](#) on page 121.



In HPOM, you can use Perl functions only. C functions do *not* work.

### Example:

The `getTeamInfo` function uses the Message Group attribute as a parameter and returns a string containing the name of the Team responsible for the event.

If functions return more than a single value, individual elements can be accessed by using the built-in `getByIndex` function. For example, a `myFunction` function returns the values 10, 20, and 30. These values are bound to a `myVariable` variable. To access the individual elements, you would use the built-in `getByIndex` function. For more information about built-in functions, see [Appendix A, Built-In Functions](#).

## Combine

Variable that combines two or more variables.

### Example:

You define the following variables:

```
a constant 'Hello'  
b constant 'World'  
c constant 'Rate is'  
d constant 10  
e constant 20
```

You combine the variables you defined:

```
Combine [a b] results in "Hello World".  
Combine [c <AlarmCnt>] results in "Rate is <AlarmCnt>".  
Combine [10, 20] results in 1020.
```

When an integer and a string are combined, the resulting output is a string.



## Lookup

Data returned from a Data Store lookup. The Data Store contains a table of global values. Unique keys identify each table entry. You use these keys to select a value from the table.

You can use the `Lookup` operator to query values from a Data Store and bind them to variables. Parameters for this operator are one or more variables. The values referred to by these variables are concatenated. The resulting value is used as the key in the Data Store lookup.

Each line of the Data Store file uses the following syntax:

```
ADD DATA(keyValue, ReturnValue)
```

This syntax contains the following values:

keyValue

Must be an integer or string.

ReturnValue

Can be any data type.

The Data Store file can contain multiple lines.

The first line in the file must be the header with the following format:

```
#path#date#version#0
```

You begin the comment with two hyphens (--).

Example:

The Data Store is loaded and contains one entry:

```
ADD DATA("Overheated", 80)
```

An `X` variable has a value of "Overheated". If you use `X` as a parameter of `Lookup`, a value of 80 is returned.

Two variables, `Y` and `Z`, with values of "Over" and "heated", respectively, result in a key value of "Overheated". The value 80 is returned.



Typically, the Data Store contains static topological information. For example, you could run scripts once a day to create the Data Store file and update the ECS engine with the newly created file.

For more information about the Data Store, see [Accessing External Data](#) on page 157.

## Variable Evaluation

By default, a variable is evaluated by a function when it is used. To override this default behavior, you can select when the variable is evaluated.

Variables can be evaluated at the following times:

### Default

Variable is evaluated when it is used.

### Event In

Variable is evaluated when the event participates in the correlator, after having passed the primary and secondary filter conditions.

### Correlator Creation

Variable is evaluated when the correlator is instantiated. For example, in Repeated correlation, the variable is evaluated when the correlator is instantiated. For background information on correlator instantiation, see [Message Key](#) on page 52.

### Correlator Deletion

Variable is evaluated when the instance of the correlator is deleted.

For more information about creating and deleting correlators, see [Table 1](#) on page 53. You should define all parameters (other than the standard attributes displayed in the shortcut menu) as variables.



A variable is evaluated *only once*. For example, if a variable has been flagged to be evaluated at `Event In`, but the variable is used in the Advance Filter, the variable is evaluated when the Advanced Filter is processed. The variable is *not* re-evaluated when the event enters the correlator.

Functions flagged for evaluation at `Event In`, `Correlator Creation`, and `Correlator Deletion` are *always* invoked synchronously.

An asynchronous 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 asynchronous function.

## Automatic Variables

In addition to the standard event attributes and the user-defined variables, Composer maintains the following automatic variables:

### AlarmCnt

The number of alarms that enter the correlator. For example, if you are using Rate correlation, the attribute `AlarmCnt` counts the number of alarms arriving.

You can access the variable when creating new alarms or defining callback functions in the New Alarm and CallBacks tabs, respectively.

### CorrelationDuration

The actual time taken to apply the correlator. For example, if you are using Rate correlation, you might generate a new alarm when the rate exceeds 5 in 30 minutes. If the rate is breached in the tenth minute, the variable is bound to the value 10.

## Advanced Filter

After filtering events with an Alarm Signature (primary filter), you can further filter the events in the correlator, based on the Advanced Filter (secondary filter) condition. The Advanced Filter is a set of data structures consisting of Name, Operator, and Value fields. For a list of supported attributes and operators, see [Alarm Signature](#) on page 43.

For example, if a `Router_Failure` alarm is received from a Core router, there is a requirement to generate a new alarm. By examining the event attributes only, you cannot determine whether the alarm is emitted from a Core router. To solve this problem, you might define an `isCoreRouter` variable that is bound to the return value of a `GetIsRouter` function. This function takes the agent-address as its parameter. If the router is a Core router, it returns 1. Otherwise, it returns 0. In the Advanced Filter, you define a correlator that checks whether `isCoreRouter` is set to 1. In this way, you make sure that the alarm is applied only to Core routers.

## Message Key

A defined correlator is merely a template to indicate the interaction of alarms.

For example, you might define a Multi-Source correlator with two alarms: `router_down` and `interface_down`. The `router_down` alarm suppresses individual `interface_down` alarms emitted from the same router. If a router fails, the correlator discards individual interface (component) failures from the same router.

The specified Multi-Source correlator is generic in the sense that it applies to all `router_down` and `interface_down` alarms. However, an `interface_down` alarm should be suppressed only if the router to which the interface belongs has also failed. The mechanism that ties alarms together is the message key. The message key is evaluated when the alarm enters the correlator. Alarms that match the value of the message key are correlated together.

In the Multi-Source correlator you defined, you can use the name of the router as a message key if you can extract the router name from both the `router_down` and the `interface_down` alarms. The message key can be a physical entity (for example, an interface or a router) or a logical entity (for example, a service or customer).

When an alarm enters the correlator, the event has passed the primary (Alarm Signature) and secondary (Advanced Filter) filters for the correlator, and the message key is evaluated. If there is no instance of the correlator with the evaluated message key, an instance of the correlator is created with the message key. Creating this new instance is called “correlator creation.” If an instance of the correlator exists for the same message key when the event arrives, the incoming alarm is correlated under the correlator for this message key. In other words, the alarm is correlated with the other alarms that are evaluated with the same message key. The message key is necessary only when two or more alarms must be related. For example, in Suppress correlation, there is no requirement for a message key because the correlator is applied to all alarms that meet the Alarm Signature and Advanced Filter conditions.



Do not confuse “message key” in this section with Message Key attributes or Message Key Correlation attributes, which are defined in HPOM policies. In this section, “message key” simply refers to an event’s attribute. If multiple events (as defined by their signatures) contain the same value in the message key attribute, they are considered “linked” for the correlator. However, each alarm can reference a different event attribute. As long as the values are identical, everything is fine.

The point at which an instance of the correlator is deleted is dictated by the semantics of the correlation model, as shown in [Table 1](#).

**Table 1 Correlator Creation and Deletion**

Correlation Model	Requires Message Key	Operational Mode	Point of Correlator	
			Creation	Deletion
Enhance	No	N/A	N/A	N/A
Multi-Source	Yes	Mode 1	At the first alarm with a message key for which no correlator has been instantiated	When the time window for the last event for this message key expires
		Mode 2	At the first alarm with a message key for which no correlator has been instantiated	<ul style="list-style-type: none"> <li>• When the set is complete</li> <li>• When the set is not complete (time)</li> <li>• After the last alarm</li> <li>• When the message key enters the system</li> </ul>
Rate	Yes	N/A	At the first alarm with a message key for which no correlator has been instantiated	<ul style="list-style-type: none"> <li>• When the rate is breached</li> <li>• When the time window expires from the time the last event for this message key enters the system</li> </ul>
Repeated	Yes	N/A	At the first alarm with a message key for which no correlator has been instantiated	After correlator instantiation
Suppress	No	N/A	N/A	N/A

**Table 1 Correlator Creation and Deletion**

Correlation Model	Requires Message Key	Operational Mode	Point of Correlator	
			Creation	Deletion
Transient	Yes	No threshold specified	When the first Fail alarm arrives with a message key for which no correlator has been instantiated	<ul style="list-style-type: none"><li>• When a Clear alarm arrives within the time window of the Fail alarm (immediately on getting a pair)</li><li>• When the time window expires without a Clear alarm arriving (no pair detected)</li></ul>
	N/A	Threshold specified	N/A	<ul style="list-style-type: none"><li>• When the threshold number of pairs is received in the specified time window (at the point the threshold is breached)</li><li>• When the threshold is not breached and the threshold time window is after the last pair was created</li></ul>
User-Defined	N/A	N/A	N/A	N/A

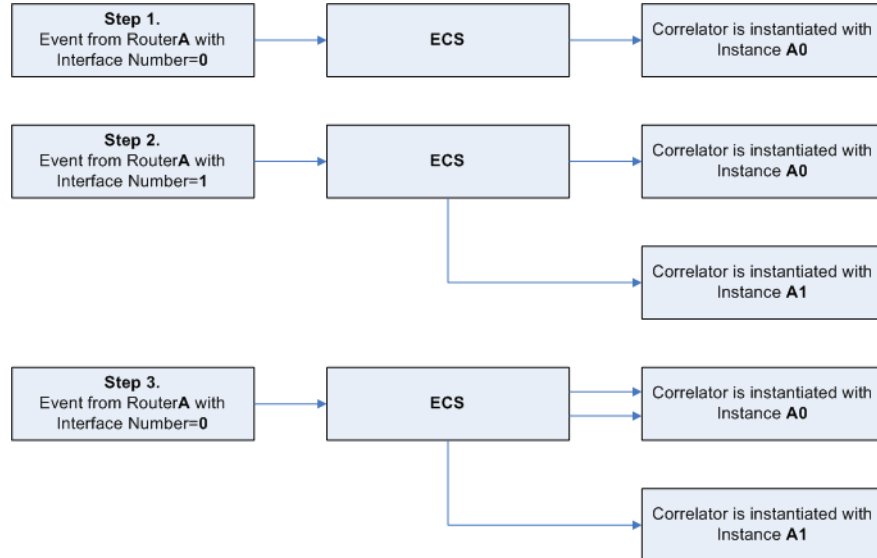
### Example 1: Generating New Router Alarms

You have a requirement to generate a new alarm when the number of alarms received from the same router is higher than 20 within one hour. All events matching the Alarm Signature are categorized as router alarms. To create a mechanism to examine alarms emitted from the same router, you could assign the agent-address as the message key.

## Example 2. Monitoring Interface Failure Rates

You have a requirement to monitor the rate at which interfaces on a router are failing. You could use the `IF_Rate` correlator, in which the message key is `x+<Interface Number>` (for example, `x+varbind0`). In this correlator, `x` could be an attribute, as shown in [Figure 7](#).

**Figure 7 Example of Message Key**



## Parameters

To change the default behavior of the basic correlator type, you set parameters. The functionality of the parameters can vary across the different correlator templates. For details about parameter functionality, see [Chapter 7, Use Cases in NNM](#) and [Chapter 8, Use Cases in HPOM](#).

## New Alarm Tab

In the Correlator Configuration window, the New Alarm tab enables you to define correlators that generate new alarms, as shown in [Figure 8](#). For example, you might use Repeated correlation to generate an alarm that reports the number of alarms that arrive in a specified time window.

**Figure 8** New Alarm Tab

Multi-Source

Name:

Description Definition **New Alarm** CallBacks

New Alarm Specification

Alarm No1

**New Alarm Definition**

Name	Value
APPLICATION	swap_util.APPLICATION
MSGTEXT	swap_util.msg_text
OBJECT	swap_util.msg_obj
SEVERITY	swap_util.SEVERITY
GROUP	swap_util.GROUP
SERVICE_NAME	swap_util.SERVICE_NAME

New Previous Next Delete

Feedback

OK Cancel Help

When you create and alter alarms, you can enrich them. Enriched alarm information includes text from multiple alarms, customer details, affected users and systems, and so on.




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



To create a new alarm, you can use one of the following options:

- **New Alarm Specification**

Create a new alarm with all new attributes. The New Alarm Specification includes Name and Value. The New Alarm Specification displays all of the mandatory attributes that must be entered to create a new alarm.

 To find out how to create a new alarm, see [Creating a New Alarm](#) on page 97. To find out how to add event attributes to the `CO.conf` file, see [Adding Attributes](#) on page 72.

In addition, you can feed the alarm back into the circuit. If you want the new event to be fed back into the system, where it can participate in other correlators, click the **Feedback** button.

- **Alter Specification**

Create a new alarm by altering some of the attributes of the existing alarm:

Field

Field that must be altered. The drop-down menu displays all attributes for the selected event type. For more information about event attributes, see [Appendix B, Event Attributes](#).

Mode

Mode used to alter event attributes:

Replace


New value replaces the event attribute of the original alarm.

Append

New value is appended to the existing event attribute.

Value

Value of event attributes appended or replaced with new values.

 The Alter Specification tab is *not* 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, when you use Transient correlation, the attributes displayed are always that of the last Clear alarm.

## Callbacks Tab

In the Correlator Configuration window, the Callbacks tab enables you to execute a function after it has participated in correlation, as shown in [Figure 9](#). When you create a new alarm, you can invoke an external command to perform user-specific functions (for example, logging or issuing trouble tickets).

**Figure 9 Callbacks Tab**

Enhance

Name

Description Definition New Alarm Callbacks

Create Discard

Function Name

Function Description

Function Type: Perl

Function Usage: Default

☒ Synchronous ☐ Asynchronous

**Parameter List**

No.	Parameters
1	

Function Definition

OK Cancel Help

You can write the external function in C or Perl. Or you can use built-in functions that enable you to select the parameters for the function from the Parameter List table. The location of the C or the Perl file containing the external function is specified in Composer.



In HPOM, you can use Perl functions only. C functions do *not* work.

To find out how to write C functions, see [Writing C Functions](#) on page 104. To find out how to write Perl functions, see [Writing Perl Functions](#) on page 113.

You can run callback functions by using the following tabs:

- **Create tab**

Runs the Callback function (if defined) when a new alarm is *created*.

- **Discard tab**

Runs the Callback function (if defined) when the alarm is *discarded*.

You use the Create or Discard tab to define when the Callback function is run.

## Composer Menus

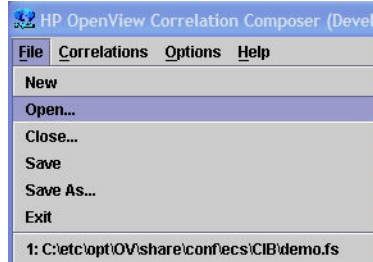
The Composer GUI includes the following menus:

- [File Menu](#) on page 60
- [Correlations Menu](#) on page 61
- [Options Menu](#) on page 62
- [Help Menu](#) on page 63

## File Menu

The Composer GUI includes a File menu, as shown in [Figure 10](#).

**Figure 10 File Menu**



The File menu contains the following items:

### New

Creates an empty Correlator Store file. If an existing Correlator Store file is still open, Composer closes the file. If you have not yet saved the file, you are prompted to save it.



A Correlator Store is the same as an ECS Fact Store.

### Open

Displays the file browser, which enables you to find and open a previously saved Correlator Store.

### Close

Closes the opened Correlator Store file. If you have not yet saved the file, you are prompted to save it.

### Save

Saves the Correlator Store file. This item is disabled if the Correlator Store file has not been modified since the last save.

### Save As

Saves the Correlator Store file to a different name. Selecting this item displays the file browser in which you enter the new file name.

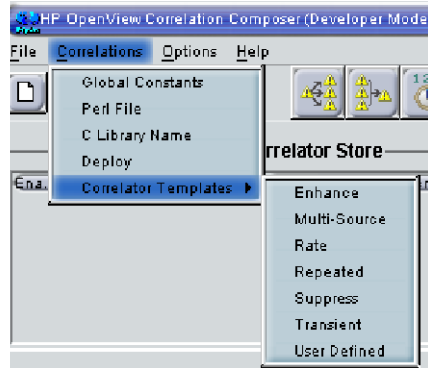
### Exit

Closes the Correlator Store file and exits Composer. If the file has unsaved changes, you are prompted to save the file before exiting.

## Correlations Menu

The Composer GUI includes a Correlations menu, as shown in [Figure 11](#).

**Figure 11 Correlations Menu**



The Correlations menu contains the following items:

### 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, in which you specify the Perl file that contains all of the functions.

### C Library Name

*NNM only:* Displays the C Library Name window, in which you specify the C library name.

### Deploy

*NNM only:* Deploys the Correlator Store files to the HP Event Correlation Services (ECS) engine. This option is enabled only when Composer is started in Operator mode. For more information about the ECS engine, see [ECS Engine](#) on page 141.



In HPOM environments, you merge and deploy by using the `ovocomposer` command. For details, see [Merging and Deploying Correlator Store Files](#) on page 154.

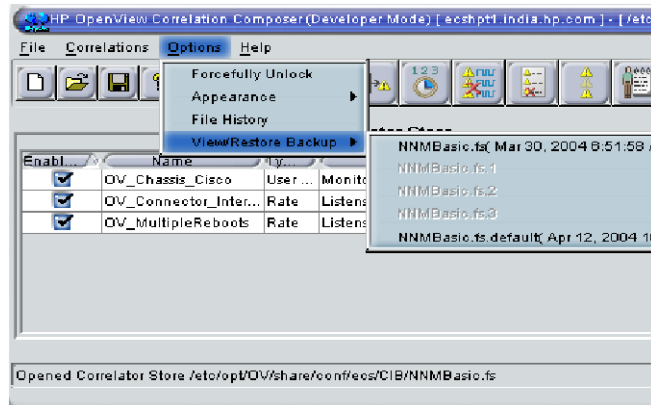
### Correlator Templates

Displays a submenu from which you can select the kind of correlation you want.

## Options Menu

The Composer GUI includes an Options menu, as shown in [Figure 12](#).

**Figure 12 Options Menu**



The Options menu contains the following items:

### Forcefully Unlock

Provides mutually exclusive access to the Correlator Store. For details, see [Locking Files](#) on page 261.

### Appearance

Displays a submenu from which you can select the look and feel of the interface.

### File History

Displays the File History window that displays a list of recently opened files.

### View/Restore Backup

Displays a submenu from which you can select the version of a backed-up file that you want. For details, see [Restoring Backed-Up Files](#) on page 77.

## Help Menu

The Composer GUI includes a Help menu, as shown in [Figure 13](#).

**Figure 13 Help Menu**



The Help menu contains the following items:

### **Overview**

Displays the online help for Composer.

### **Table of Contents**

Displays the table of contents for the online help. You can also view the online help index from this window.

### **About**

Displays the current release and copyright information for Composer and associated software.

# Composer Toolbars

Composer includes the following toolbars:

- [Standard Toolbar](#) on page 64
- [Correlator Templates Toolbar](#) on page 65
- [Deploy Button \(NNM Only\)](#) on page 66

## Standard Toolbar

The standard toolbar buttons, shown in [Figure 14](#), provide shortcuts to frequently used menu commands.

**Figure 14 Standard Toolbar Buttons**



The standard toolbar contains the following buttons:



**New**

Creates a new Correlator Store. Displays the Input Event Type window from which you can select the input event type.



**Open**

Displays a file browser from which you can 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 contents of the online help for Composer.



## Correlator Templates Toolbar

The correlator templates toolbar, shown in [Figure 15](#), provides shortcuts to the various correlator templates.

**Figure 15 Correlator Templates Buttons**



The correlator templates toolbar contains the following buttons:



### **Enhance**

Displays the Enhance Correlation window. From this window, you can define parameters to add more information to an alarm before the alarm is output.



### **Multi-Source**

Displays the Multi-Source Correlation window. From this window, you can define related or sympathetic events, and discard or output events with enriched information.



### **Rate**

Displays the Rate Correlation window. From this window, you can define parameters to maintain a count of event arrival and output a new event based on this count.



### **Repeated**

Displays the Repeated Correlation window. From this window, you can define parameters to discard events of similar type.



### **Suppress**

Displays the Suppress Correlation window. From this window, you can define parameters to discard a certain class of events.



### **Transient**

Displays the Transient Correlation window. From this window, you can define parameters and correlate based on some threshold values.



### **User-Defined**

Displays the User-Defined Correlation window. From this window, you can define correlators based on your requirements.

## Deploy Button (NNM Only)

The Deploy button provides a shortcut to the “Deploy the Correlator Stores to the ECS” engine. This button is used in NNM environments only.



### **Deploy**

Merges the Correlator Stores and loads them into the ECS engine. To find out how to deploy the Correlator Store, see [Deploying the Correlator Store](#) on page 257.



In HPOM environments, you merge and deploy by using the `ovocomposer` command. For details, see [Merging and Deploying Correlator Store Files](#) on page 154.

## 3 Getting Started

This chapter explains how to get started with HP Correlation Composer:

- [Starting Composer](#) on page 68
- [Stopping Composer](#) on page 69
- [Configuring the Correlator Store](#) on page 69
- [Defining Event Attributes](#) on page 70
- [Backing Up Files](#) on page 73



This chapter assumes you are running Composer in Developer mode. For details, see [Developer Mode \(HPOM and NNM\)](#) on page 30.

# Starting Composer

How you start Composer depends on the environment in which you invoke it.

## Starting Composer from NNM

In HP Network Node Manager (NNM), you can start Composer in HP Event Correlation Services (ECS) or from the command line.

To start Composer from NNM, do one of the following:

- **ECS**

From the ECS Configuration Management GUI, follow these steps:

- a Select the row with Composer.
- b Click **Modify**.

Composer starts in Operator mode.

- **Command Line**

From the command line, enter one of the following:

— *Operator mode*

```
ovcomposer -m o
```

— *Developer mode*

```
ovcomposer -m d
```

For details, see the *ovcomposer* reference page.

## Starting Composer from HPOM

In HP Operations Manager (HPOM), you can start Composer from the command line or from the GUI. The HPOM for UNIX GUI is a Java application that requires X-Server to display on the client.

To start Composer from the command line, enter the following:

```
ovocomposer -ui
```

Composer starts in Developer mode. (There is no Operator mode.)

For more usage options, see [Merging and Deploying Correlator Store Files](#) on page 154 as well as the *ovocomposer* reference page.

# Stopping Composer

You can stop Composer in one of two ways:

- To close the Correlator Store file, click **File→Close**. This step closes only the Correlator Store that is currently being configured.
- To exit the current Composer session, click **File→Exit**.

## Configuring the Correlator Store

Before you use the Correlator Store, make sure you fully understand your requirement. After you document your requirement, you can map it to an off-the-shelf template.

To configure a Correlator Store, follow these steps:

### 1 Document your requirement.

To make sure you fully understand your correlation requirement, document it in distinct steps. Make sure to document the set of alarms that participates in the correlation.

For example, if the requirement is to discard `node_down` alarms from routers, you can document how to recognize a `node_alarm` from the router. A `node_down` alarm contains attributes that determine whether the alarm is a `node_alarm`. Typically, you recognize a `node_alarm` in the Alarm Signature section. Examine the attributes to determine whether the `node_down` alarm is from a router. If there is no attribute within the alarm to indicate that it is from a router, you recognize it in the Advanced Filter.

### 2 Map the requirement to a correlator template.

For descriptions of correlator templates, see [Correlator Templates](#) on page 31. For scenarios in which multiple correlator templates are used to meet a requirement, see [Chapter 9, Developer Mode in NNM](#).

# Defining Event Attributes

Composer ships with standard event attributes. In HPOM and NNM, you can add attributes by editing the `CO.conf` file.

## Default Attributes

The default configuration shipped with Composer contains standard event attributes, defined by event type (SNMP for NNM, OpC for HPOM). You can configure additional attributes by editing the `CO.conf` configuration file in the default directory where Composer is installed, as listed in [Table 2](#).

**Table 2    Directories for Composer and CO.conf**

Operating System	Composer Installation	Configuration File
HP-UX	/opt/OV/bin	/opt/OV/bin
Linux	/opt/OV/bin	/opt/OV/bin
Solaris	/opt/OV/bin	/opt/OV/bin
Windows	%OvInstallDir%\bin	%OvInstallDir%\bin




To make changes to the configuration file, you need root or administrator user access.

## Changing Mandatory Attributes

If you create a new alarm in the New Alarm tab of Composer, you must set values for mandatory fields. The list of mandatory attributes are specified in the `CO.conf` file. You can reduce the number of attributes you must set when using Create New Alarm specification.

To change the mandatory attributes in HPOM, follow these steps:

- 1 Open the `CO.conf` file.
- 2 In the `##MANDATORYFIELDS` section, search for `#OpC_Msg`.  
This is the first occurrence of the string `"OpC_Msg"`.
- 3 Change the mandatory fields:
  - Remove fields that you do not want to be mandatory.
  - Add fields (for example, CMAs) that you want to be mandatory. In HPOM, `MSGTEXT` and `OBJECT` are mandatory fields.
- 4 Save and close the file.
- 5 Restart Composer.

## Adding Attributes

In HPOM and NNM, you can add attributes by editing the `CO.conf` file.

To add attributes, follow these steps:

- 1 Open the `CO.conf` file in any standard text editor.

To find out where this file is located on your platform, see [Table 2](#) on page 70.

- 2 Edit the `CO.conf` file.

Examples:

- *HPOM*

You might want to add a custom message attribute (CMA) named `Customer-ID` to the file.

To get started, search for the second occurrence of the string `"OpC_Msg"` and add `Customer-ID` to a new line:

```
#OpC_Msg
AACTION_ACK
AACTION_ANNOTATE
AACTION_CALL
AACTION_NODE
```

- *NNM*

The default `CO.conf` file contains variable bindings from 0 to 12 (100 in later releases).

To add a variable binding, search for the second occurrence of the string `"SNMP"` in this file and add the following:

```
varBind[13]->name
varBind[13]->value
```

- 3 Save and close the file.
- 4 Restart Composer.



# Backing Up Files

Composer enables you to recover from a disaster by making regular backups of Correlator Store files automatically.

## Automatic Backups of Correlator Store Files

When you create or open a file for the first time, the Correlator Store creates a default backup file (if it does not exist already). This file remains constant throughout the life of the Correlator Store.

The file contains the changes made to the Correlator Store in the current session. The backup file is identified by a numerical extension, which indicates the version. When you save the file for the first time in the first session, the file name is appended with an extension of .1 (for example, filename.1). This number is incremented each time you save the file for the first time in subsequent sessions (for example, filename.2).

You can edit backed-up files by user and group only. [Table 3](#) lists the file permissions for the Correlator Store and its backed-up files.

**Table 3    File Permissions for the Correlator Store and Backed-Up Files**

File Type	HP-UX, Linux, Solaris		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

## Example of Backed-Up Files

An example of a backed-up file in the Correlator Store is the `ATM.fs` file. Every time you open a new session of `ATM.fs` and save it the first time, the backup files roll to the next version number.



This example assumes that users can view only three backup versions. To find out how to increase the number of backed-up files, see [Overriding the Number of Backed-Up Files](#) on page 76.

The Correlator Store backs up the `ATM.fs` file as follows:

### 1 Creating the file creates the default backup (`ATM.fs.default`).

When you create or open the Correlator Store for the first time, it creates a backup `ATM.fs.default` file.

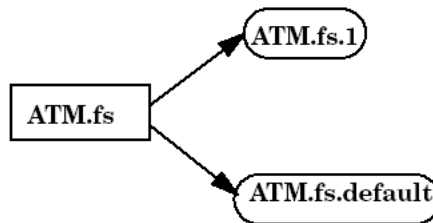
This file remains constant throughout the life of `ATM.fs`. Every time you edit and save `ATM.fs`, the changes are reflected in `ATM.fs`.

### 2 Editing the file for the first time creates `ATM.fs.1`.

When you open the *first* session of `ATM.fs` file and save it the first time, the following occurs:

- a Contents of the existing `ATM.fs` are saved as `ATM.fs.1`.
- b Your edits are written to `ATM.fs`.

Session 1

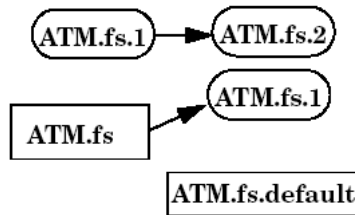


### 3 Editing the file for the second time creates **ATM.fs.2**.

When you open the *second* session of `ATM.fs` and save it the first time, the following occurs:

- a Contents of the existing `ATM.fs.1` file are saved as `ATM.fs.2`.
- b Contents of the existing `ATM.fs` file are saved as `ATM.fs.1`
- c Your edits overwrite `ATM.fs`.

#### Session 2

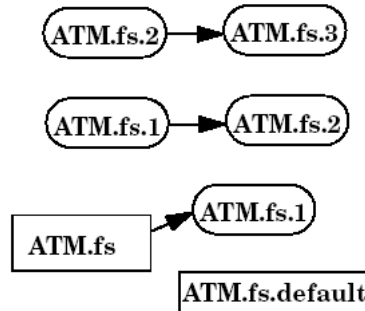


### 4 Editing the file for the third time creates **ATM.fs.3**.

When you open the *third* session of `ATM.fs` and save it the first time, the following occurs:

- a Contents of the existing `ATM.fs.2` file are saved as `ATM.fs.3`.
- b Contents of the existing `ATM.fs.1` file are saved as `ATM.fs.2`.
- c Contents of the existing `ATM.fs` file are saved as `ATM.fs.1`.
- d Your edits overwrite `ATM.fs`.

#### Session 3



## 5 Editing the file for the fourth time overwrites previous backups.

When you open the *fourth* session of `ATM.fs` and save it the first time, you begin to overwrite previous backups:

- a Contents of the existing `ATM.fs.2` file overwrite `ATM.fs.3`.
- b Contents of the existing `ATM.fs.1` file overwrite `ATM.fs.2`.
- c Contents of the existing `ATM.fs` file overwrite `ATM.fs.1`.
- d Your edits overwrite `ATM.fs`.

The backed-up file `ATM.fs.3` from the third session is now updated with new contents. To prevent this from happening, you can increase the number of backed-up files. For details, see [Overriding the Number of Backed-Up Files](#) on page 76.

## Overriding the Number of Backed-Up Files

By default, three backed-up versions of any given file in Composer are visible to users. You can override the number of backed-up files by editing the `MAX_BACKUP` field in the `CO.conf` configuration file.

To override the number of backed-up files visible to users, follow these steps:

- 1 Open the `CO.conf` file in any standard text editor.  
To find out where this file is located on your platform, see [Table 2](#) on page 70.
- 2 In the `CO.conf` file, replace the default number (3) under the heading `MAX_BACKUP` by the number of backup files you want users to view.  
For a Correlator Store, the maximum number of backups allowed is 20.
- 3 Save and close the file.
- 4 Restart Composer.

## Restoring Backed-Up Files

You can open, view, and restore backed-up versions of files in Composer.



This procedure assumes that users can view only three backup versions. To find out how to increase the number of backed-up files, see [Overriding the Number of Backed-Up Files](#) on page 76.

To restore backed-up files, follow these steps:

- 1 To open a backed-up file, click **Options**→**View Backup**→**<FileVersion>**.

The following message displays:

You are now viewing an archive version of the file.

- 2 To restore the archive version you are viewing, click **Save**.

This step makes the backup file you are viewing the latest version of the file. Previously backed-up files roll down to accommodate the change.



When you revert changes to the latest file, you overwrite the previously saved version of the file you are viewing. Make sure that you are not overwriting data erroneously.



---

## 4 Developing Correlators

This chapter explains how to develop Correlator Stores in HP Correlation Composer:

- [Developing Correlator Stores](#) on page 80
- [Configuring Correlator Stores](#) on page 83
- [Managing Correlators](#) on page 102
- [Writing C Functions](#) on page 104
- [Writing Perl Functions](#) on page 113
- [User-Defined Correlation](#) on page 117
- [Merging Correlator Store Files](#) on page 124


# Developing Correlator Stores

This section explains how to create, open, modify, and migrate Correlator Stores in Composer.

## Creating a Correlator Store

To create a new Correlator Store file, follow these steps:

- 1 To open a new Correlator Store file, do one of the following:

- In the Composer main menu, click **File→New**.
- In the standard toolbar, click the  **New** icon.

If a file is already open, Composer prompts you to save the file.



The default event type is `SNMP`. To find out how to change the event type, see [Defining Event Types](#) on page 84.

- 2 From the main menu, select **File→Save**.
- 3 In the File panel, enter the Correlator Store file name.


Use file names that start with a letter and contain only letters, numbers, and underscores (`_`). For example, `my_configuration` is a valid file name. The extension `.fs` is supplied automatically.

- 4 Click **OK** to save the Correlator Store file.

You can save the Correlator Store file under any directory. Before saving the file, make sure the path you specified is correct.

## Opening a Correlator Store

To open a Correlator Store file, follow these steps:

- 1 To select the file to open, do one of the following:
  - From the file browser window, click **File→Open**.
  - In the standard toolbar, click the  **Open** icon.

The Open File browser window displays.



- 2 Select the name of the file you want to open.
- 3 Click **Open**.

The Correlator Store file displays.



Correlator Stores created in Composer versions before Composer 3.3 must be migrated to the latest version. To find out how to migrate Correlator Stores, see [Migrating a Correlator Store to Composer 3.3](#) on page 82.

## Modifying a Correlator Store

After you create a Correlator Store, as described in [Creating a Correlator Store](#) on page 80, you can modify its properties, as needed.

To modify the Correlator Store, follow these steps:

- 1 Select **File**→**Open** to open the Correlator Store.  
The file browser window opens.
- 2 From the file browser window, select the file name.  
The Correlator Store with the correlators displays.
- 3 To open a correlator, do one of the following:
  - Double-click the correlator.
  - Right-click the correlator in the table and click **Modify**.The Correlator window opens.
- 4 Edit the Correlator Store file.
- 5 From the main menu, select one of the following:
  - Click **File**→**Save** to save the file to its current file name.
  - Click **File**→**Save As** to save the file to another file name.



In NNM, only Save and Exit options are available. NNM has a default Correlator Store. All updates must be made to this file.

## Migrating a Correlator Store to Composer 3.3

You *must* migrate Correlator Stores created before Composer 3.3 to the latest version with the following script:

- *UNIX*

```
$OV_CONTRIB/ecs/csmigrate.ovpl
```

- *Windows*

```
%OvInstallDir%\contrib\ecs\csmigrate.ovpl
```

To migrate a Correlator Store to Composer 3.3, enter the following:

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

This command includes the following parameters:

*<Correlator Store name>*

Name of the Correlator Store to be migrated.

*<ENGLISH|JAPANESE|CHINESE>*

Native language of the Correlator Store to be migrated.

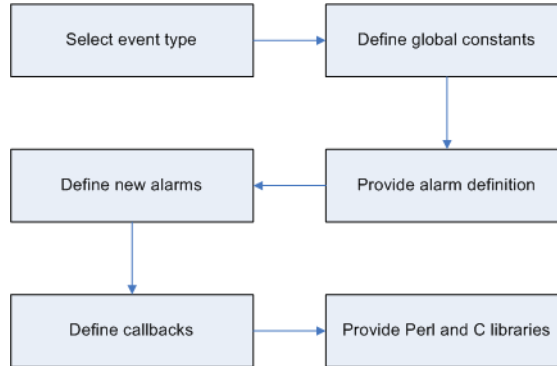
*<final Correlator Store name>*

Name of the Correlator Store after migration.

# Configuring Correlator Stores

Figure 16 shows the tasks you perform when configuring a Correlator Store.

**Figure 16 Planning the Configuration**



To configure a Configuration Store, perform the following tasks:

- Task 1: [Defining Event Types](#) on page 84
- Task 2: [Optional: Defining Global Constants](#) on page 85
- Task 3: [Defining Alarm Correlators](#) on page 87
- Task 4: [Optional: Defining New Alarms](#) on page 96
- Task 5: [Optional: Creating Callback Functions](#) on page 99
- Task 6: [Setting the Perl File Location](#) on page 101



Although tasks 2, 4, and 5 are optional, it is recommended that you perform them if you have to define any of the parameters described in the tasks.

## Defining Event Types

The event type determines the kind of events that enter HP Event Correlation Services (ECS).

Composer supports the following event types:

### **HPOM**

HPOM messages. Used only with HPOM.

### **SNMP**

Default. SNMP traps. Used only for NNM.

You select the event type when you create a Correlator Store.



The default event type is *SNMP*. You can create a Correlator Store for one event type at a time. If you want to change the event type, close the currently opened Correlator Store and repeat the procedure.

# Optional: Defining Global Constants

Composer enables you to define global constants that bind values to names. You can then refer to the values by these names when defining correlators.

## Value Types for Global Constants

When you define named value pairs, the parameters you enter can refer to global constants. For example, a named value called `MULTI_LOGIN_FAIL` that you have defined can have the event attribute `"multi_login_failure"`.

Table 4 lists examples of value types for global constants.

**Table 4    Examples of Value Types for Global Constants**

Value Type	Example
Integer	123
Float	123.45
String	"1234"
	'abcd'
OID	1.2.3.4

## Defining a Global Constant

To define a global constant, follow these steps:

- 1 From the main menu, click **Correlations**→**Global Constants**.

The Global Constants Definition window opens.



- 2 Enter the following data:

### Name

Name of the constant.

### Value

Value for the name.

These values can be referenced anywhere inside the Correlator Store.



To add more global constants, right-click a row and click **Add** from the shortcut menu. A new row is then added to the Global Constants Definition table.

- 3 Click **OK** to close the window.

## Deleting a Global Constant

To delete a global constant, follow these steps:

- 1 From the main menu, click **Correlations**→**Global Constants**.

The Global Constants Definition window opens.

- 2 For each global constant you want to delete, do one of the following:
  - Right-click the global constant and click **Delete** from the shortcut menu.
  - Click the global constant and press **DELETE**.
- 3 Click **OK** to close the window.

## Defining Alarm Correlators

To define alarm correlators, perform the following tasks:

- Task 1: [Creating a Correlator](#) on page 87
- Task 2: [Defining a Correlator](#) on page 87
- Task 3: [Defining Variable Types](#) on page 90

### Creating a Correlator

Every correlator has a unique name that identifies it, a description that states briefly what the correlator is expected to do, and a definition.

To create a new correlator, follow these steps:

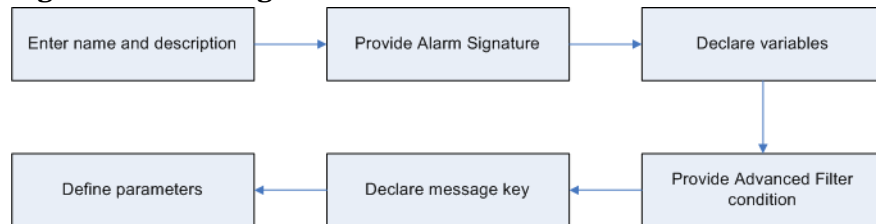
- 1 Click **Correlation**→**Correlation Templates**.
- 2 From the list of templates, select the type of correlator you want to create.  
The Correlator window opens.
- 3 Define the correlator.

For instructions, see [Defining a Correlator](#) on page 87.

### Defining a Correlator

[Figure 17](#) summarizes the steps you follow to define a correlator.

**Figure 17 Defining a Correlator**



To define a correlator, follow these steps:

**1 Enter a name for the correlator.**

In the Correlator window, enter a name in the Name text box.

The alarm is referenced by this name throughout the Correlator Store. Use names that start with a letter and contain only letters, numbers, and underscores (\_). Special characters (for example, !, @, #, ^, &, and \*) are not allowed. Make sure the name indicates the problem type (for example, Generator\_OFF).

**2 Enter a description of the alarm.**

Briefly state the cause of the alarm and what the correlator does.

**3 Define the Alarm Signature.**

To add an Alarm Signature, follow these steps:

**a** Click the **Definition** tab.

The Alarm Definition panel displays.

**b** Define the Alarm Signature.

The Alarm Signature is a set of values that specifies a filter.

**c** From the drop-down list, select the **Field** name.

For a list of valid event attributes for the event types supported by Composer, see [Appendix B, Event Attributes](#).

**d** From the drop-down list, select the **Operator** value.

For a list of valid operators, see [Alarm Signature](#) on page 43.

**e** Enter the value of the field for which the Alarm Signature is described:

— To enter a value, click the **Value** cell and type the value.

— To select a global constant, double click the **Value** cell.

A shortcut menu displays the previously defined constants. Choose the constant from the menu.



#### 4 **Declare the variables.**


Variables are names with associated values that can be used inside the correlator definition.

To declare a variable, follow these steps:

- a Enter a name for the variable.
- b From the drop-down menu, select the variable type.  
For a list of variable types supported, see [Variables](#) on page 47.
- c Depending on the variable type you chose, enter a value.  
To enter the value, click the **Value** cell. For instructions, see [Defining Variable Types](#) on page 90.

#### 5 **Define the message key.**

To define the message key, follow these steps:

- a Click the **Message Key** box.  
A shortcut menu displays all possible values.
- b Select the variable or attribute you want to use as the message key.  
 You do not need to define a message key for Suppress, Enhance, or User-Defined correlations.

#### 6 **Optional: Define the Advanced Filter.**

From the shortcut menu in the respective columns, select the Attributes, Operator, and Value.

This field is non-editable. For values to display, they must be previously defined.

#### 7 **Define the parameters.**

For descriptions of all the buttons in the parameters section, see [Chapter 7, Use Cases in NNM](#) and [Chapter 8, Use Cases in HPOM](#).

#### 8 **Create the correlator.**

Click **OK** to complete the creation of the correlator.

## Defining Variable Types

This section explains how to define variable types:


- [Defining Constant Values](#) on page 90
- [Combining Variables](#) on page 90
- [Extracting Value Patterns](#) on page 92
- [Defining Functions](#) on page 93
- [Validating Function Definitions](#) on page 95

### Defining Constant Values

You use constant values for reference when defining correlators.

To define a constant value, follow these steps:


- 1 Click the **Value** cell.
- 2 Enter the value.

 In NNM, object IDs do not have the leading dot. For example, 1.2.3.4 is valid, but .1.2.3.4 is not.

The value displays in the cell.

### Combining Variables

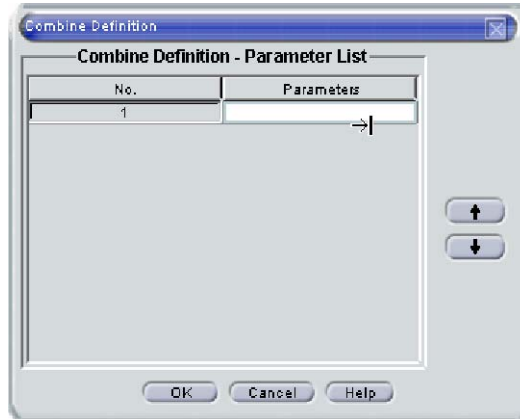
You can define a new variable by combining two or more previously defined variables, attributes, or global constants.

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

To combine the values of two or more variables, follow these steps:

- 1 Click the **Value** cell.

The Combine Definition window opens.



- 2 Click the **Parameters** cell.



Before you combine variables, make sure the Parameters column contains parameters for them. If the column does not contain parameters, you cannot combine the variables.

A shortcut menu displays all attributes, predefined variables, and global constants.

- 3 From the shortcut menu, select the attribute or variables.

The selected item displays in the Parameters cell.

- 4 To add a variable or attribute to the list, right-click the list and select **Add** from the shortcut menu.

A new row is added.

- 5 *Optional:* To view all attributes and variables from a shortcut menu, click the **Parameters** cell.

- 6 Select the variables, attributes, or global constants that you want to combine.

Repeat this step to combine more variables, attributes, or global constants.

- 7 Click on **OK** to close the Combine Definition window.

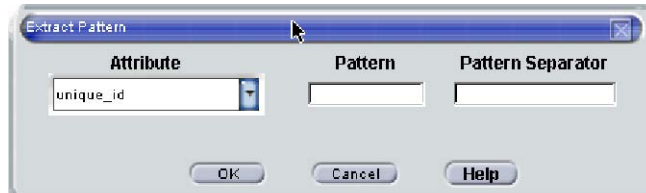
## Extracting Value Patterns

You can extract and use event attribute values inside correlation definition.

To define an extract pattern, follow these steps:

- 1 From the drop-down menu, select **Extract**.
- 2 Click the **Value** cell.

The Extract Pattern window opens.



- 3 Select the attribute from which you want to extract a substring.
- 4 In the Pattern text box, enter the pattern you want to extract.  
For examples of extract patterns, see [Appendix C, Pattern Matching](#).
- 5 In the Pattern Separator text box, enter the pattern separator.  
By default, the pattern separator is an empty space.
- 6 Click **OK** to close the Extract Pattern window.

## Defining Functions

The variable type can be a function whose return value is bound to the name of the variable. To find out how to write functions in C, see [Writing C Functions](#) on page 104. To find out how to write functions in Perl, see [Writing Perl Functions](#) on page 113.



To create a Callback function to be used when discarding the event, perform this procedure in the Discard window instead of the Function window.

To define a function, follow these steps:

- 1 Click the Function tab.

The Function window opens.



The Function window is a dialog box with a title bar labeled "Function". It contains several input fields and controls:

- Function Name:** A text input field with a dropdown arrow on the right.
- Function Description:** A text input field with a button to its right.
- Function Type:** A dropdown menu currently showing "Builtin".
- Function Usage:** A dropdown menu currently showing "Default".
- Synchronous/Asynchronous:** Two radio buttons. "Synchronous" is selected.
- Parameter List:** A table with two columns: "No." and "Parameters". It contains one row with "1" in the "No." column and an empty "Parameters" column. To the right of the table are up and down arrow buttons.
- Function Definition:** A large text area for writing the function code, with a button to its right.
- Buttons:** "OK", "Cancel", and "Help" buttons at the bottom.

- 2 From the Function Type drop-down menu, select one of the following function types:

- **C**
- **Perl**
- **Built-in**

Perl and C functions are external functions that you supply. Built-in functions are packaged with Composer.

Based on the type you select, the appropriate function names are loaded in the Function Name field. To find out how to enable loading of C or Perl function names, see [Configuring the UserDevelopedFuncDetails.xml File](#) on page 111.

- 3 In the Function Name field, select or enter the name of the function you want to call.



As a prefix to the function name, you can add the name of the C library name in which the function resides. For example, if a function named `BSCName()` resides in a library called `SNMPlib`, you can type the function name as **SNMPlib:BSCName()** in the Function Name text box.

Likewise, as a prefix to the function name, you can add the name of the Perl library name in which the function resides. For example, for a function named `function1` in a Perl file named `User1.pm`, you can enter **User1::function1**. For details, see [Support for Multiple Perl Files](#) on page 115.

- 4 In the Function Description field, type a brief description of the function.  
To view the entire description window, click the ellipses (...) button.
- 5 From the Function Usage drop-down menu, select the phase at which the external function has to be invoked:

- **Default**

External function (written in Perl or C) is called.

- **Event In**

Function is called when the event enters ECS.


- **Correlator Creation**

Function is called when the correlator is instantiated.

- **Correlator Deletion**

Function is called when the instance of the correlator is deleted.

For details about correlator creation and deletion, see [Table 1](#) on page 53.

- 6 To select the mode in which the function must be called, click one of the following:
  - **Synchronously**
  - **Asynchronously** These options are not valid for Composer built-in functions, which are always invoked synchronously.
- 7 From the shortcut menu in the Parameter list, add the parameters for the function.
- 8 Click **OK** to complete the function definition.

### Validating Function Definitions

The details you enter in the Function window are validated against XML files:

- **Built-in functions**

The built-in functions are validated with the data in the following file:

- *UNIX*

`$OV_CONF/ecs/CIB/BuiltInFuncDetails.xml`

- *Windows*

`%OvInstallDir%\conf\ecs\CIB\BuiltInFuncDetails.xml`

Do *not* edit this file. The built-in functions provided with ECS are *not* extensible.

- **C and Perl functions**

The C and Perl functions are validated with the data in the following file:

- *UNIX*

`$OV_CONF/ecs/CIB/UserDevelopedFuncDetails.xml`

- *Windows*

`%OvInstallDir%\conf\ecs\CIB\UserDevelopedFuncDetails.xml`

This file is extensible. You can add details about the C and Perl functions that have been developed to this file. For instructions, see [Configuring the UserDevelopedFuncDetails.xml File](#) on page 111.

- **Number of parameters**

For function definitions with a *fixed* number of parameters, Composer validates the number of parameters. For function definitions with a *variable* number of parameters, Composer verifies that the definitions have a minimum number of parameters. For example, the `StoreStr()` function can hold a variable number of parameters even though the minimum number of parameters is five. If the number of parameters does not match the expected value, an error message appears. You can then make the required modifications to the function definition.

The data type of parameters is not validated.

## Optional: Defining New Alarms

You can define new alarms in one of two ways:

- [Changing Alarm Attributes](#) on page 96
- [Creating a New Alarm](#) on page 97

### Changing Alarm Attributes

You can create new alarms by altering the attributes of existing alarms.

To change the attributes of an alarm, follow these steps:

- 1 In the Correlator window, click the **New Alarms** tab.  
The New Alarms panel opens. By default, the None option is selected.
- 2 From the drop-down menu, select **Alter Specification**.



The Alter Alarm Definition table displays.

Rate

Name

Description Definition **New Alarm** Callbacks


Alter Specification

**Alter Alarm Definition**

Field	Mode	Value

☐ Feedback

OK Cancel Help

- 3 From the drop-down menu in the Field column, select the attribute you want to change.
  - 4 From the drop-down menu in the Mode column, select the mode of alteration you want to use:
    - **replace**  
Replaces the existing event contents.
    - **append**  
Appends the new values to the existing contents of the attribute.
  - 5 From the shortcut menu, select the variable that is to be replaced or added to the attribute contents.
-  Any new values to be appended must already be defined in the Variables section of the correlator.

## Creating a New Alarm

You can create new alarms for outgoing events.

To create a new alarm, follow these steps:

- 1 From the drop-down menu in the New Alarms panel, select **New Alarm Specification**.

The New Alarm Definition table displays the mandatory attributes.

Multi-Source

Name:

Description Definition **New Alarm** CallBacks

New Alarm Specification

Alarm No1

Name	Value
APPLICATION	swap_util.APPLICATION
MSGTEXT	swap_util.msg_text
OBJECT	swap_util.msg_obj
SEVERITY	swap_util.SEVERITY
GROUP	swap_util.GROUP
SERVICE_NAME	swap_util.SERVICE_NAME

- 2 From the shortcut menu, select values for the fields.

➤ Names are OpC\_Msg message attributes. Values are case-sensitive, user-defined variables.

The list of attributes are based on the configuration file:

```
<Composer install directory>/CO.conf.
```

If you need to add more attributes, you must edit the CO.conf file. For instructions, see [Defining Event Attributes](#) on page 70. To find out how to change mandatory attributes, see [Changing Mandatory Attributes](#) on page 71.

- 3 To add new attributes to the new alarm, right-click the row and select **Add**.

A new row is added to the New Alarm Definition table. For a description of the buttons in this table, see [New Alarm Definition Table](#) on page 98.

The text Alarm No. 1 displays in the upper-right corner of the table. You can navigate through the list of alarms defined.

## New Alarm Definition Table

In correlators, the New Alarm Definition table contains the following buttons:

### Delete

Deletes the new alarm.

### Feedback

Sends all alarms back to Composer.

**New**

Creates a new alarm and displays a New Alarm Definition table.

**Next**

Moves to the next alarm created. Displays the contents of the succeeding alarm in the new alarms list.

**Previous**

Moves to the previous alarm created. Displays the contents of preceding alarm in the new alarms list.

## Optional: Creating Callback Functions

When a correlator discards or creates an alarm, you can invoke a user-defined function by using C, Perl, or built-in functions. You can choose parameters for these functions from the set of variables defined for the correlator. Typically, you use the Callback functions to create audit trails. For example, when you delete an event, you can invoke a logging function.

To create a Callback function, follow these steps:

- 1 In the Correlator window, click the **CallBacks** tab.  
The Callbacks panel displays.
- 2 From the Function Name drop-down menu, select the name of the Callback function.
- 3 In Function Description field, type a description that describes what the function does.
- 4 From the Function Type drop-down menu, enter the mode in which the function must be called.
- 5 From the Function Usage drop-down menu, select the time at which the function is to be called.
- 6 From the shortcut menu in the Parameters table, select the parameters to the external function.
- 7 *Optional:* To add more parameters to the function, right-click and select the attributes from the shortcut menu.
- 8 Click **OK** to complete the Callback function definition.

## Callback Variables

You can access all alarms attributes by the corresponding alarm names.

### Callback Functions

You can access automatic variables by using the following functions:

- **Create Callback**

You can use the Create Callback function to access the attributes of the new alarm just created by using the New Alarm automatic variable.

- **Discard Callback**

The automatic variables for the Discard Callback function depend on the correlation model you choose.

### Automatic Variables

Automatic variables are available for specific correlation models only:

- **Discarded** (Multi-Source model only)

In the Multi-Source model, you can access the attributes of the discarded alarm from the Discarded automatic variable. You can also access the attributes of the alarms in the set by name.

- **Suppressor** (Transient model only)

In the Transient model, the Discard Callback function is called only for the Fail alarm. The Callback function can access all attributes of the Clear alarm by using the Suppressor automatic variable.



No automatic variables are available for the Enhance, Rate, Repeated, Suppress, or User-Defined modes.

## Setting the Perl File Location

Before you can write external functions in Perl, you must provide Composer with the Perl file location.

To set the Perl file location in Composer, follow these steps:

- 1 In the Correlator Store window, click **Correlations**→**Perl File**.

The Perl File window opens.

- 2 Enter the name of the Perl file.

By default, the Perl file is located in the following directory:

- *UNIX (HP-UX, Linux, and Solaris)*  
`$OV_CONTRIB/ecs/external/perl`
- *Windows managed node*  
`%OvInstallDir%\contrib\ecs\external\perl`
- *Windows management server*  
`%OvShareDir%\contrib\ecs\external\perl`

To set Perl files in a different location, you must specify the relative path in the Perl File window. To find out how to reference multiple Perl files, see [Writing Perl Functions](#) on page 113.

- 3 Click **OK** to close the Perl File window.

# Managing Correlators

This section explains how to do the following:

- [Opening a Correlator](#) on page 102
- [Modifying a Correlator](#) on page 103
- [Deleting a Correlator](#) on page 103



In the Operator mode, Composer does not allow you to create new correlators. For details, see [Chapter 10, Operator Mode in NNM](#).

## Opening a Correlator

To open an existing correlator, follow these steps:

- 1 From the main menu, click **File**→**Open**.  
The file browser displays all the Correlator Stores that have been defined.
- 2 Do one of the following:
  - In the File panel, enter the Correlator Store name.
  - From the directory listing, select the file name.The Correlator Store displays all existing correlators.
- 3 Double-click the correlator you want to open.  
The Correlator window opens.

## Modifying a Correlator

After you define a correlator, you can modify its properties, as needed.

To modify a correlator, follow these steps:

- 1 In the Correlator Store window, do one of the following:
  - Double-click the name of the correlator you want to open.
  - Right-click the name of the correlator you want to open, and then select **Modify** from the shortcut menu.

The Correlator window opens.

- 2 Modify the properties of the correlator.
- 3 Click **OK** to save your changes.

The Correlator window closes. You return to the Correlator Store window.

## Deleting a Correlator

When you delete a correlator, the selected correlation record is deleted from the Correlator Store file. Composer sorts the remaining correlators by name in alphabetical order.

To delete a correlator, follow these steps:

- 1 Open the Correlator Store window.
- 2 Do one of the following:
  - Select the correlator you want to delete, and then press **Delete**.
  - Right-click the correlator you want to delete, and then select **Delete** from the shortcut menu.

# Writing C Functions

This section describes how to write external C functions for Composer.

## Creating a C Function

To create a C function that is accessible in Composer, follow these steps:

- 1 Write the C function.

Follow these guidelines:

- [Skeleton Code for C Functions](#) on page 105
- [Signatures for C Functions](#) on page 107
- [Parts of C Functions](#) on page 107

- 2 Create a shared library of the C function in the following location:

— *UNIX*

`$OV_CONTRIB/ecs/external`

— *Windows*

`%OvInstallDir%\contrib\ecs\external`

For more information about C libraries, see [Managing Correlators](#) on page 102.

- 3 Configure the `UserDevelopedFuncDetails.xml` file for ease of use in Composer GUI.

For instructions, see [Configuring the UserDevelopedFuncDetails.xml File](#) on page 111.



## Skeleton Code for C Functions

When writing C functions, use the following skeleton code:

```
#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 responsibility 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;
}

```

## Signatures for C Functions

From Composer, invoke the following signature for all C functions:

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

This signature contains the following parameters:

`argc`

Number of arguments passed.

`argv`

Array of pointers to the arguments.

`reqId|cmdId`

Opaque parameters used when running the Callback function.

`callback`

Pointer to a function must be invoked on completion. The function passes back any return values to Composer.

## Parts of C Functions

C functions consist of the following parts:

- Part 1: [Passing Arguments](#) on page 107
- Part 2: [Processing Arguments](#) on page 108
- Part 3: [Returning Values](#) on page 108

### Passing Arguments

You begin a C function by passing arguments from Composer. The function can take any number of arguments of any type. To ensure that the correct parameters are passed, you must configure Composer correctly.

In [Skeleton Code for C Functions](#) on page 105, the function assumes three parameters: an integer, a string, and an object ID. The pointers to these parameters are in `argv`. They are accessed as `argv[0]`, `argv[1]`, `argv[2]`. When accessing these pointers, make sure they are cast as the right type.

## Processing Arguments

After passing arguments from Composer, the C function processes them. The C function *must not* free the parameters that it has passed from Composer. When it is evoked, the Callback function frees space.

**Table 5** lists the data type that are passed from and received by Composer.

**Table 5     Data Types in Composer**

Passed from Composer	Received by Composer
Integer	Integer
Float	Float
String	Char *
Object ID	Char *
Time	Integer

## Returning Values

After processing arguments from Composer, the C function returns a value or set of values back to Composer. The C function indicates that processing has completed by invoking the Callback function. This mechanism enables you to process either synchronously or asynchronously. In other words, the function returning does not indicate function completion.

You can extract the arguments, queue them up for processing by some other thread, and return immediately. After processing, you can invoke the Callback function to return the values to Composer and simultaneously indicate completion. (The call to the Callback function is made from a function other than the function that was originally called.)

Extracting arguments in this way is useful when, for example, the function must go over the network to access data, or when databases must be accessed, both of which take time. If the function being invoked takes very little time, it is recommended that you call the Callback function and then return it.

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



The Callback function must be called even if the function succeeds.

Returning values consists of four tasks:

- Task 1: [Allocating Space for the Return Values](#) on page 109
- Task 2: [Wrapping the Return Values](#) on page 109
- Task 3: [Marshalling the Return Values](#) on page 110
- Task 4: [Running the Callback Function](#) on page 111

### Allocating Space for the Return Values

To allocate space for return values, the function makes the following calls:

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

In this command, X is the number of return values that must be returned.

The retValue call is defined as follows:

```
- GC_Values ** retValue = NULL;
```

### Wrapping the Return Values

The GC\_MAKEVALUE macro is contained in the GC\_Values.h file in the following directory:

- *UNIX (HP-UX, Solaris, and Linux)*

```
$OV_HEADER/ecs/ECS
```

- *Windows*

```
%OV_HEADER%\ecs\ECS
```

To wrap return values, the function invokes the following macro:

```
GC_MAKEVALUE(GC_INTEGER|GC_STRING|GC_OLD|GC_FLOAT, valToReturn,  
strVal);
```

```
GC_Values * strVal = NULL
```

This macro contains the following parameters:

`GC_INTEGER`

Returns an integer.

`GC_STRING`

Returns a string.

`GC_OID`

Returns an object ID. To return an Object ID, the `valToReturn` parameter should be a string in the dot notation.

Example: "1.2.3.4"

`GC_FLOAT`

Returns a float.

`valToReturn`

Holds the value to be returned.

`strVal`

Points the macro `GC_Values` (for example, `GC_Values * strVal;`).

### Marshalling the Return Values

After allocating space and wrapping returning values, the function combines the two.

Assuming there are two values to be returned, the function calls the following:

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

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

This call includes the following:

`retValue`

Return value whose space is defined in [Allocating Space for the Return Values](#) on page 109.

`valToReturn1 | valToReturn2`

Returned values, which are wrapped in [Wrapping the Return Values](#) on page 109.

## Running the Callback Function

After combining allocated space and wrapped return values, the function indicates that the process is complete and returns the values.

The function calls the following:

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

This call includes the following:

`callback`

Points to a function that was passed as the fifth parameter.

`reqId` and `cmdId`

Third and fourth parameters passed.

## Configuring the UserDevelopedFuncDetails.xml File

To eliminate the task of manually entering function details (for example, name, signature, and call mode) when creating a function definition, you can configure the `UserDevelopedFuncDetails.xml` file. This file enables you to access function details quickly and easily from the Composer GUI (for example, in the Function Definition dialog box).



Composer supports this feature only in Developer mode. If this feature is not used, you can manually enter the function details as before.

The XML file is located in the following directory:

- *UNIX*

```
$OV_CONF/ecs/CIB/UserDevelopedFuncDetails.xml
```

- *Windows*

```
%OV_CONF%\ecs\CIB\UserDevelopedFuncDetails.xml
```

To configure the `UserDevelopedFuncDetails.xml` file, follow these steps:

- 1 For each function, add the following details to the XML file:

`FunctionName`

Name of the function in the following format:

*LibraryName:FunctionName*

`FunctionDescription`

Brief description of the function behavior.

FunctionSignature

Signature of the function signature.

Example:

```
add int1 int2
```

In this example, *int1 int2* are the integer values to be added.

No\_Of\_Args

Number of parameters to be added. -1 indicates a variable number of parameters.

Min\_No\_Of\_Args

Minimum number of parameters expected. Applicable only if the number of parameters is -1.

FunctionType

Type of function. Value can be C or Perl.

FunctionUsage

Usage of the function.

Value can be one of the following:

- Default
- Correlator Creation
- Correlator Deletion
- Correlator EventIn
- Function call mode

FunctionCall

Function call. Value can be Asynchronous or Synchronous.

## 2 Validate the updated XML file.

When Composer starts, it validates the XML file for syntax and semantic errors. The function definitions are validated with the corresponding XML schema file, *function.xsd*, located in the same directory as the XML file. If errors are found, they are reported on the command line or in a log file (if XPL is enabled). If no errors are found, the parser loads the definitions into memory.



# Writing Perl Functions

This section describes how to write external Perl functions for Composer.

## Creating a Perl Function

To create a Perl function that is accessible in Composer, follow these steps:

- 1 Write the Perl function.

For guidelines, see [Skeleton Code for Perl Functions](#) on page 113.



It is strongly recommended that you test Perl functions outside of Composer. The embedded Perl interpreter is known to exit on syntax or parse errors.

- 2 Store the file in the following location:

- *UNIX*

```
$OV_CONTRIB/ecs/external/perl
```

- *Windows managed node*

```
%OvInstallDir%\contrib\ecs\external\perl
```

- *Windows management server*

```
%OvDataDir%\contrib\ecs\external\perl
```



Only one Perl file can be loaded into the Composer at run time. If multiple Perl files are required, see [Support for Multiple Perl Files](#) on page 115.

## Skeleton Code for Perl Functions

This section includes skeleton code you can use to write Perl functions. In this skeleton, the `orchPerlfunction` function contains the key elements required to write a Perl function. The arguments are passed to the function as an array. Individual parameters can be accessed as array elements. Object IDs are passed as a string in the dot notation format (for example, "1.2.3.4").

Multiple return values can be returned. If an object ID must be returned, it must be encapsulated before returning. If you need to return an object ID, include the `ecd1Encap` subroutine.

```
# Function to encapsulate return values into ECS specific data types
# Do not modify
sub ecd1Encap
{
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");
# ecd1Encap(\@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
# ecd1Encap(\@retVal0, "ECS_OID");
# return(\@retVal0, $retVal1, $retVal2);
# return a Object ID, int and a string
}
```

## Support for Multiple Perl Files

Although Composer supports multiple Perl files, it does not support multiple Perl files with more than one `MAIN` routine (`BEGIN` block).

### Including Files on UNIX

On UNIX, to include one Perl file in another Perl file, you need the `use` keyword:

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

t.pl
BEGIN{
    push(@INC, '\\opt\\OV\\contrib\\ecs\\external\\perl\\');
}
use p;
```

In this example, the Perl file `t.pl` includes the file `p.pm` by using the keyword `use`. Only `t.pl` includes a `BEGIN` block.



On UNIX, Perl files are normally stored in the following directory:

```
/opt/OV/contrib/ecs/external/perl/
```

If Perl files are stored in this directory, you do not need to provide the complete path when loading Perl to the ECS engine.

### Including Files on Windows

On Windows, to include one Perl file in another Perl file, you use the following:

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

t.pl
BEGIN{
    push(@INC, $ENV{OvInstallDir} . 'contrib\\ecs\\external\\perl\\');
};
```

Only `t.pl` includes a `BEGIN` block.



On Windows, Perl files are normally stored in the following directory:

- *Managed node*  
`%OvInstallDir%\contrib\ecs\external\perl`
- *Management server*  
`%OvDataDir%\contrib\ecs\external\perl`

If Perl files are stored in this directory, you do not need to provide the complete path when loading Perl to the ECS engine.

## Creating a Main Perl File

When creating multiple Perl files, it is recommended that you create one main Perl file that contains a `BEGIN` block that performs a few initialization tasks, at most. Make sure to include all the other Perl files in the main Perl file by adding a `use` keyword.

For example, two users create individual Perl files. The first user develops a set of Perl functions in `user1.pm`. The second user develops a set of Perl functions in `user2.pm`. You can include both files in the main Perl file, `main.ovpl`, by using the statements `use user1;` and `use user2;`. Only the `main.ovpl` file must be referenced from Composer.

In this example, if the files `user1.pm` and `user2.pm` are in a different location than the `main.ovpl` file, the file `main.ovpl` must add their locations:

```
push(@INC, <location of Perl files>);
```

If `user1.pm` and `user2.pm` have clashing function names, the `main.ovpl` file must use the scope resolution operator (`::`) when making a function call like `user1::f()`.



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

To eliminate the task of manually entering function details whenever you create a function definition in Composer, you can configure the `UserDevelopedFuncDetails.xml` file. For details, see [Configuring the UserDevelopedFuncDetails.xml File](#) on page 111.

# User-Defined Correlation

If the predefined correlator templates do not meet your requirement, you can use the User-Defined template. Like the other templates, the User-Defined template has Alarm Definition, New Alarm, and Callback sections.

## Input Functions

In the User-Defined template, you call the input function when the event satisfies the Alarm Signature and Advanced Filters. You can write the input function in Perl, C, or built-in functions (for example, `makelist`).

The return value must be in the following format:

*flag, window, alarm mask, optional values*

This return format includes the following parameters:

*flag*

Mandatory. Action to be taken on the alarm. You can provide multiple values with OR.

This parameter can have one or more of the following values:

ALTER

Alters the event, as specified in the Alter Alarm specification.

Default: 4

CREATE

Creates a new event, as specified in the New Alarm specification.

Default: 16

DISCARD

Discards the event.

Default: 2

## HOLD

Holds the event for the time window specified in the *window* parameter. After this time window, it invokes the specified output function. If you specify the parameter, the event is held for the specified time window, regardless of other correlators outputting the same event. At the specified time, the output function is invoked. Its return value determines the action to be taken on the event.

Default: 1

## PASSTHROUGH

Outputs the event. The event is output *only* if no other correlator is used to DISCARD or HOLD it.

Default: 8

## WEAKHOLD

Holds the event for the time window specified in the *window* parameter. If other correlators output the event, a copy of the event is held, and the output function is invoked after the number of seconds specified in *window*.

Typically, you use WEAKHOLD to invoke the output function at the end of the specified time window to send a new alarm (by creating or altering the alarm).

Default: 64

## PSEUDOHOLD

Does not hold the event. However, it calls the output function after the number of seconds specified in *window*. If you specify PSEUDOHOLD, the output function cannot return ALTER or CREATE. If you need to create or alter alarms in the output function, you must use either WEAKHOLD or HOLD. Use PSEUDOHOLD when the output function needs to be called after the specified time window (typically, to do cleanup).

Default: 32



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

*window*

**Mandatory.** Valid only if the `flag` parameter contain `HOLD`, `WEAKHOLD`, or `PSEUDOHOLD`. This parameter indicates the time, in seconds, after which the output function is invoked.

*alarm mask*

**Mandatory.** Controls the set of alarms to be created. In the New Alarm section, several alarms can be defined. However, in many cases, only a subset of these new alarms must be created. You use this parameter to control the set of alarms to be created. For example, if five alarms have been defined in the New Alarm section, and this parameter is set to 3 (last two bits are set), the first and second alarms are created.



This parameter is valid only if you choose the `CREATE` option as part of the *flag* parameter. Otherwise, the value is ignored.

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

*optional values*

**Optional.** Any values other than the mandatory values returned are bound to a variable called `InputRetVal`. For example, if the function returns only one optional value (for example, `HOLD, 5, 10`), the `InputRetVal` variable is bound to the value 10, and can be used like any other variable. If the function returns more than one optional value (for example, `HOLD, 5, 10, 20, 30`), the `InputRetVal` variable is bound to a list that contains more than one optional value (for example, 10, 20, and 30). You can access individual elements by using the built-in `getByIndex` function.

## Output Functions

In the User-Defined template, you invoke output functions when the *flag* returned by the input function is either HOLD, PSEUDOHOLD, or WEAKHOLD. The function is called after the number of seconds specified in the *window* parameter of the input function.

The return value must be in the following format:

*flag, alarm mask, optional values*

This return format includes the following parameters:

*flag*

Mandatory. Action to be taken on the alarm.

*alarm mask*

Mandatory. Controls the set of alarms to be created. For details, see [Input Functions](#) on page 117.

*optional values*

Optional. Additional values returned by the function. These values are bound to an OutPutRetVal variable. You access the OutPutRetVal variable the way you access the InputRetVal variable. For details, see [Input Functions](#) on page 117.



## Writing a User-Defined Function

To write input or output functions in C for User-Defined correlation, you use the process for writing external C or Perl functions:

- [Writing C Functions](#) on page 104
- [Writing Perl Functions](#) on page 113

The only differences are the return values.

### Return Values

User-Defined correlation uses the following return values:

- **Input function**  
Must return at least two values: flag and window period.
- **Output function**  
Must return at least one value: flag.

### Flag Values

You define the values for the flags in the `GC_Values.h` file:

- *UNIX (HP-UX, Solaris, and Linux)*  
`$OV_HEADER/ecs/ECS/GC_Values.h`
- *Windows*  
`%OV_HEADER%\ecs\ECS\GC_Values.h`



Although environment variables (for example, `%OV_CONTRIB%` and `%OV_HEADER%`) exist on Windows, they are not set globally. Instead, the variables are set by the `"%OvInstallDir%\bin\ov.envvars.bat"` script, which must be executed before Composer can be launched.

To set the required environment variables, enter the following:

```
"%OvInstallDir%\bin\ov.envvars.bat"
```

## Skeleton Code for User-Defined Functions

When writing input or output functions in C for User-Defined correlation, use the following skeleton code:

```
#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

To merge two Correlator Stores, you use the `csmerge` tool. For example, you may need to merge two Correlator Stores when you update a correlator in your production environment with the latest revision of the correlator. Likewise, you may need to update a correlator when you add newly developed correlators to your production environment.

► In HPOM environments, you merge and deploy by using the `ovocomposer` command. For details, see [Merging and Deploying Correlator Store Files](#) on page 154.

Within a Correlator Store, no two global constants or correlators can have the same name. When two global constants or correlators have the same name, but different values or correlation logic, they clash. If there is a clash, external input is required to continue with the merge. You provide external input interactively or by specifying them in the configuration file. If there is no clash in names, the merge is automatic.

The `csmerge` tool is available in the following directory:

- *UNIX (HP-UX, Solaris, and Linux)*

`$OV_CONTRIB/ecs`

- *Windows*

`%OvInstallDir%\contrib\ecs`

► The `csmerge` tool is implemented as a Perl script. It requires a Perl 5.6 or higher. The `%OV_CONTRIB%` environment variable is valid on Windows only if `ov.envvars.bat` is sourced.

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 execute only one command at a time. The `csmerge` command ignores all commands except the first.

## Merging Correlator Stores Specified in the NameSpace File

You can merge Correlators Stores listed in a given namespace 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. If there is an overlap of names of global constants, the global constants are prefixed with the logical name of the Correlator Store. For this reason, it is important that the logical names for correlators be unique.

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

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

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

This command includes the following parameters:

*<Namespace filename>*

Name of the NameSpace file from which the Correlator Store file is picked.

*<final Correlator Store name>*

Name of the merged Correlator Store.

Example:

```
csmerge.pl -namespace /etc/opt/OV/share/conf/ecs/CIB/  
Namespace.conf /tmp/demoFactstore.fs
```

## Removing User Descriptions from the Correlator Store

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

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

This command includes the following parameters:

*<Correlator Store name>*

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

*<destination Correlator Store name>*

Name of the Correlator Store without the user description.

## Merging Correlator Stores

You can merge Correlator Stores that are created, but not listed in the NameSpace configuration file, to a single Correlator Store.

To merge two Correlator Stores, type the following:

**csmerge <file1> <file2> <mergedfile> -config <configuration filename>**

This command contains the following parameters:

*<file1>* and *<file2>*

Correlator Stores to be merged.

*<mergedfile>*

File that results from the merger.

*<configuration filename>*

File that specifies which values are considered while merging the Correlator Stores. If you specify this option, the user is *not* prompted for input, and all specifications are taken from the configuration file.

If there is a clash, one of following things can happen:

- Definition is picked from File1.
- Definition is picked from File2.
- Both definitions are picked, but the name for one of them is changed.

In interactive mode (where there is no configuration file), users must choose one of these three options. In non-interactive mode, the configuration file resolves the clash.

## Configuration File

The configuration file has the following format:

`decision-tag:<list of comma-separated names>`

The decision-tag variable is one the following:

`Global_Constant_from_File1`

Global Constant from File1 is used in the mergedfile.

`Global_Constant_from_File2`

Global Constant from File2 is used in the mergedfile.

`Correlator_from_File1`

Correlator from File1 is used in the mergedfile.

`Correlator_from_File2`

Correlator from File2 is used in the mergedfile.

When creating a configuration file, you must follow these punctuation rules:

- Separate names with commas.
- Begin every decision-tag on a new line.
- Place the decision-tag and the list of names on the same line.
- Separate the decision-tag and the list of names with a colon (:).
- Place comments on new lines that begin with a number sign (#). Each line that begins with a number sign in the first column is read as a comment.

For an example of a configuration file, see [Example 2: Configuration File](#) on page 129.

## Example 1: Clashing Global Constants

As an example of clashing global constants, `File1` has a global constant called `Clash` with a value of 100. `File2` has a global constant called `Clash` with a value of 200.

This conflict can result from two possible scenarios:

- **Intention**

The value of the global variable was deliberately changed to 200, in which case the only global that is in the merged file is `Clash` with a value of 200.

In this case, the configuration file would look like this:

```
Global_Constants_from_File2:Clash
```

- **Coincidence**

The names being the same is merely a coincidence. In this case, you need to add both to the merged file, but change the name of one of the variables. The configuration file is empty, but the file must be present. The merge tool automatically changes one of the names, and then adds both to the merged file.



## Example 2: Configuration File

A configuration file might look like this:

```
Global_Constant_from_File1:a,b
```

```
Global_Constant_from_File2:x,y
```

```
Correlator_from_File1:i,j
```

```
Correlator_from_File2:m,n
```

This configuration file specifies the following for the merge tool:

```
Global_Constant_from_File1:a,b
```

If there is a clash of global constants a or b, the global constants a and b in the merged file are taken from File1.

```
Global_Constant_from_File2:x,y
```

If there is a clash of global constants x or y, the global constants x and y in the merged file are taken from File2.

```
Correlator_from_File1:i,j
```

If there is a clash of correlators with names i or j, the correlators i and j in the merged file are taken from File1.

```
Correlator_from_File2:m,n
```

If there is a clash of correlators with names m or n, the correlators m and n in the merged file are taken from File2.

In addition, if there is a name clash other than those specified in the configuration file, both names are used in the merged file after one of the files is renamed. The tool generates a unique name by appending an underscore (\_) to the name in File2. For example, if there is a name clash for a correlator with the name z, both correlators are added to the merged file. The correlator from File2 is renamed z\_.



The default C library and Perl filename are always taken from File2.



## 5 Composer in NNM

This chapter explains how to use the HP Correlation Composer in the HP Network Node Manager (NNM) environment:

- [Correlator Stores](#) on page 131
- [Operator Mode](#) on page 132
- [Developer Mode](#) on page 135
- [Built-In Function](#) on page 137



In the NNM environment, Composer is available through the HP Event Correlation Services (ECS) Configuration window.

### Correlator Stores

NNM provides a set of built-in Correlator Stores that enable you to maintain correlators specific to that environment. These correlators are loaded and enabled when NNM is installed. You can enable or disable them at any time from the Correlation Composer window. For more information about these correlators, see *Managing Your Network with HP Network Node Manager*.

# Operator Mode

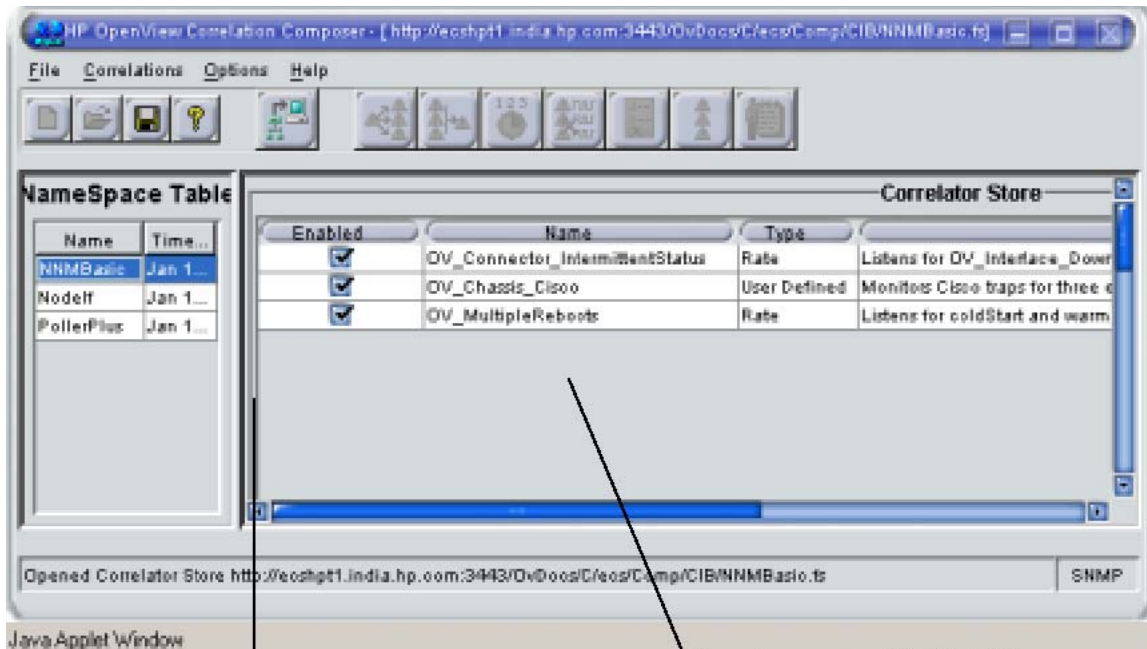
In the NNM environment, operators start Composer in Operator mode to perform basic tasks.

## Starting Composer in Operator Mode

To start Composer in Operator mode, follow these steps:

- 1 In the Event Configuration window, select **Edit→Event Correlation**.  
A web browser opens and displays correlations.
- 2 Select the row with Composer and click the **Modify** button.

The Composer window opens in Operator mode.



NNM Built-In Correlator Stores

Correlators for OV\_NodeIf

## Operator Tasks

When operators launch Composer from NNM, permissions to certain menus, correlators, and correlator fields are restricted.

In particular, operators can perform only the following tasks:

- **View correlators**

View correlators defined for the listed NameSpaces. By default, only the NNM built-in correlators are visible in the Composer GUI.

- **Edit parameters**

Edit those parameters defined as editable. By default, almost all of the correlator parameters for the NNM built-in correlators are *not* editable.

- **Save correlators**

Save changes made to the Correlators by doing one of the following:

- Click the **Save** button.
- Click **File**→**Save**.

Save Correlator Store files to the following directory:

- *UNIX*

`$OV_CONF/ecs/CIB`

- *Windows*

`%OV_CONF%\ecs\CIB`

You must deploy the Correlator Store to apply its changes to the ECS engine.

- **Deploy Correlator Stores**

Deploy Correlator Stores to the ECS engine by doing one of the following:

- Click the **Deploy** button.
- Click **Correlations**→**Deploy**.

- **Launch online help**

Launch the online help contents by doing one of the following:

- Click the **Help** button.
- Click **Help**→**Table of Contents**.

## Operator Menu Options

In the NNM environment, operators can use the following Composer menu options:

### **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**

Closes the Composer window.

### **Correlations→Global Constants**

Opens the Global Constants window, where you can edit the values of Global Constants that are declared editable in the Security file. For more information about the Security file, see [Security File](#) on page 244.

### **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 loading the merged file into the ECS engine. For details about the Deploy procedure, see [Deploying the Correlator Store](#) on page 257. To find out how to create the Deploy configuration file, see [Creating Deploy Configuration Files](#) on page 251.

### **Options→Forcefully Unlock**

Provides mutually exclusive access to the Correlator Store. For information about file locking, see [Locking Files](#) on page 261.

### **Options→Appearance**

Displays a submenu for selecting the look and feel of the interface.

### **Options→View/Restore Backup**

Displays a submenu to select the backup file version. For information about backing up files, see [Backing Up Files](#) on page 73. In the NNM environment, all users have read and write permission for backup files.

**Help→Overview**

Displays the first page of the Composer online help.

**Help→Table Of Contents**

Displays the table of contents of the Composer online help. You can also view the online help index from this window.

**Help→About Correlation Composer**

Displays the current release and copyright information for Composer and associated software.

## Developer Mode

In the NNM environment, you start Composer in Developer mode to maintain the configuration files and use built-in functions.

### Starting Composer in Developer Mode

To start Composer in Developer mode, type the following command:

```
ovcomposer -m d
```

### Configuration Files

In addition to creating correlation logic, developers maintain the configuration files required by Composer. The definitions provided in these files govern how Composer functions in the NNM environment. The default configuration files required by Composer are shipped with the product.

In the NNM environment, developers use the following configuration files:

- [NameSpace File](#) on page 136
- [Security File](#) on page 136
- [Deploy Configuration File](#) on page 137

## Namespace File

The default Namespace file resides in the following directory:

- *UNIX*

`$OV_CONF/ecs/CIB`

- *Windows*

`%OV_CONF%\ecs\CIB`

This file does not contain any entries. It must be edited by NNM users. The listing in this file governs what is displayed in the Composer window.

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 contains text such as the following:

```
#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 in which the Correlator Store is saved.

Typically, the Security file resides in or below the following directory:

- *UNIX*

`$OV_CONF/ecs/CIB`

- *Windows*

`%OV_CONF%\ecs\CIB`



For example, if you want to enable operators to edit the values of Window Period and Count for the correlator `OV_Connector_IntermittentStatus`, you edit the Security file `NNMBasic.sec`.

This NNM Basic Security file contains text such as the following:

```
#NOTE: No space between the comma separating the variable fields
#ALL_TEMPLATE=ALL_PARAMS
OV_Connector_IntermittentStatus=WINDOW, COUNT
```

## Deploy Configuration File

The default Deploy Configuration file resides in the following directory:

- *UNIX*  
`$OV_CONF/ecs/CIB`
- *Windows*  
`%OV_CONF%\ecs\CIB`

The entries in this file are the default entries required by Composer in the NNM environment.

## Built-In Function

Composer provides the `getOIDValue` build-in function to work only with SNMP traps. You can use this function only in the NNM environment.

### `getOIDValue`

#### **Syntax**

```
getOIDValue oid failValue
```

#### **Parameters**

*oid*

Name of the variable binding for which the value is to be extracted.

*failValue*

Value that is returned by the function if the retrieve fails.

#### **Description**

The `getOIDValue` function retrieves the value of the first occurrence of the corresponding name from the variable bindings.

## Examples

Consider the following trap:

```
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
}
}
}
```

Consider the following statement:

**getODValue 1.3.6.1.4.11.2.7.2.17.0 -1**

If the retrieve is successful, the statement returns the value 95. Otherwise, the function returns -1. The `oid` and `failvalue` parameters must be declared as variables in the correlator definition section. To find out how to use functions, see [Defining Functions](#) on page 93.

## 6 Composer in HPOM

This chapter explains how to use the HP Correlation Composer in the HP Operations Manager (HPOM) environment:

- [Composer GUI](#) on page 140
- [ECS Engine](#) on page 141
- [Message Correlation](#) on page 142
- [Starting the Composer GUI](#) on page 144
- [Configuring MSI in HPOM for UNIX or Linux](#) on page 145
- [Merging and Deploying Correlator Store Files](#) on page 154
- [Accessing External Data](#) on page 157

HPOM provides Composer as a standard built-in feature. This feature includes a sample Correlator Store.



In HPOM environments, you merge and deploy by using the `ovocomposer` command. For details, see [Merging and Deploying Correlator Store Files](#) on page 154.

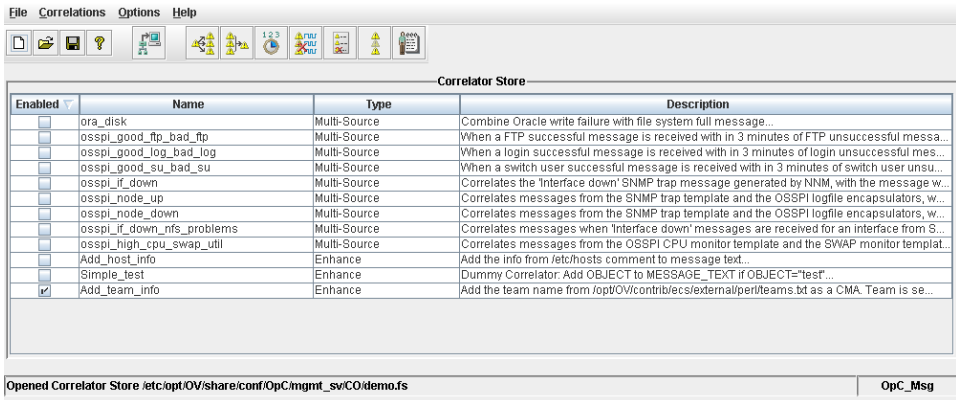


By default, *no* correlators are enabled.

# Composer GUI

The Composer graphical user interface (GUI) enables you to create and maintain all correlation logic for Composer. In HPOM, you run the Composer GUI on the HPOM management server, even if the correlators you create or maintain are destined for managed nodes (agents). The HPOM management server evaluates the local Composer startup configuration information, as shown in [Figure 18](#). For example, the HPOM management server evaluates which correlators to display, which correlator input parameters are offered, and so on.

**Figure 18 Composer GUI on the Management Server**



# ECS Engine

Composer runs inside the HP Event Correlation Services (ECS) engine, which is a component of HPOM. Composer runs as a single ECS circuit. The ECS engine correlates, suppresses, and enriches HPOM messages, based on the rules configured in deployed circuits, including Composer.

The ECS engine runs the following processes:

- **opcecm (UNIX) and OvOWECM.exe (Windows)**  
ECS on the management server
- **opceca**  
ECS on the managed nodes
- **opcecaas**  
ECS Annotation Manager Server on the management server and managed nodes

These processes are run only if they are required.

The ECS engine runs a process on the Message Stream Interface (MSI) on the management server and agent. Messages to be input to ECS must be sent to MSI. For details, see [Configuring MSI in HPOM for UNIX or Linux](#) on page 145.



In HPOM for Windows and HPOM for UNIX and Linux, the ECS circuit is called the “Event Correlation” policy type.

# Message Correlation

This section describes how to configure message correlation in the HPOM environment:

- [Correlation Options](#) on page 142
- [Correlation Guidelines](#) on page 143
- [Additional Correlation Tools](#) on page 144

## Correlation Options

When setting up Composer in the HPOM environment, consider the relative advantages of different locations for message correlation:

- **NNM domain**

You can set up event correlation in the HP Network Node Manager (NNM) domain to significantly reduce the amount of SNMP-related messages intercepted by the trap interceptor. As a rule, the earlier correlation occurs, the better. Earlier correlation reduces the load on the HPOM trap interceptor downstream. As a result, fewer messages arrive in the message browser.

- **Managed node**

Correlating messages on the managed node reduces the number of messages sent to the management server. It also reduces network traffic, database usage, and the load on the management server.

- **Management server**

Correlating messages on the management server enables you to compare and, if necessary, suppress similar or related messages coming from different managed nodes. In larger environments, you can use HPOM flexible management to configure layers of management servers into a hierarchy. In this hierarchy, the choice of where to correlate widens to include the relationship between the various levels of management server. In addition, you can correlate and enrich events arriving through HPOM Incident Web Service (IWS) on the management server.

- **Multiple sources**

You do not need to restrict correlation to individual sources. You can correlate messages from different message sources within HPOM: SNMP traps, opcmmsg, logfiles, threshold monitoring, and IWS.

For example, you might correlate the following:

- Messages generated by SNMP traps that relate to a node being down
- Messages generated by the log file encapsulator that relate to entries in a client-server application log file about an unreachable server

For a summary of correlation architecture, see [ECS Engine](#) on page 29.

## Correlation Guidelines

When correlating, pick a location that suits your plans:

- **NNM**

Correlate on NNM if you are correlating only SNMP traps together.

- **HPOM**

Correlate on the managed node if you are correlating HPOM messages that occur on the same agent.

- **Management server**

Correlate on the management server if you are correlating messages generated on different agents or if you are correlating messages arriving through IWS.

Always correlate as early as possible. Where appropriate, correlate at multiple levels. Whenever possible, suppress messages on the agent before correlating them further with messages from other agents on the managements server.

## Additional Correlation Tools

In HPOM, standard policy configurations enable you to perform common message correlation. For example, you can perform message de-duplication or simple state-based (down-up) actions without Composer. However, if the rules for state management are more sophisticated than the standard policy configurations, consider using Composer.

In HP Operations Manager i (OMi), Topology Based Event Correlation (TBEC) provides another way to do correlation.

## Starting the Composer GUI

To find out how to start the Composer GUI, see [Starting Composer from HPOM](#) on page 68.



# Configuring MSI in HPOM for UNIX or Linux

After you create correlators in the Composer GUI, and before ECS can begin to process the messages, you need to configure the Message Stream Interface (MSI). ECS can operate on the management server, the managed nodes (agents), or both.

## Configuring MSI on the HPOM for UNIX or Linux Management Server

To configure MSI on the HPOM for UNIX or Linux management server, follow these steps:

### 1 Enable output on MSI.

On the management server, type the following command:

```
/opt/OV/bin/OpC/opcsrvconfig -msi -enable [-send  
<divert|copy>]
```

When you enable output to the Server MSI, all messages are processed by ECS, even if they will never be used. This processing could negatively affect the performance of the management server.

If you choose **divert**, the messages suppressed by ECS will not be fed back into the message stream.



If you use HP Dependency Mapping Automation (DMA) or other MSI applications, you must create a `msiconf` file to serialize Composer and the MSI application.

If you choose **copy**, all original messages (before ECS processing) are already output, so suppression has no purpose. Enriched messages are provided in addition to, rather than instead of, the original messages.

The management server forwards all messages to the Server MSI. As a result, you do not need to configure specific message types to be forwarded (from the policy configuration).

### 2 Deploy the Correlation Composer policy to the server.

- a Locate and highlight the **Correlation Composer** policy.
- b In the context menu, click **Assign to Management Server**.

- c Click **Deployment**→**Deploy Server Policies**.

Alternatively, use the following commands to deploy the Correlation Composer policy to the server:

```
opcnode -assign_pol pol_name='Correlation Composer'  
pol_type=ec node_name='$MGMTSV' net_type=NETWORK_NO_NODE  
opcragt -distrib -force '$MGMTSV'
```

### 3 Verify that Composer and ECS are configured and running.

On the management server, do the following:

- Run **opcsv -status** to see these two ECS processes:

```
Event Corr. Mgr          opcecm          (14386) is running  
ECS Anno. Mgr           opcecmas       (14387) is running
```

- Run **ecsmgr -i 11 -info** to verify that the Composer circuit (ecs\_comp) is installed:

```
circuits enabled in stream - circuit ecs_comp
```

The MSI is now enabled and the ECS engine is now running.

## Configuring MSI on HPOM for UNIX or Linux Managed Nodes (Agents)

To configure MSI on HPOM for UNIX or Linux managed nodes (agents), follow these steps:

### 1 Enable output on MSI.

- a In the Node Bank, locate the managed nodes where ECS and the Composer circuit (ecs\_comp) are to be run.



The Fact Stores built in the Composer GUI can be used with the Composer circuit on the management server and managed nodes.

- b Edit the nodes and open the **Advanced** tab.
- c In the Message Stream Interface section, select **Enable Output**.
- d If you are using Composer to add actions to messages, select one or both of the following:
  - **Allow Externally Defined Automatic Actions**
  - **Allow Operator Initiated Actions**

## 2 Configure specific messages to pass to MSI.

If you chose the Divert Messages option in [step 1](#), all messages are forwarded to the Agent MSI, and this step is not required.

For each policy that you want to pass messages to MSI, follow these steps:

- a Go to the All Policies page, and locate and edit the policies that produce the messages that must be input to Composer.
- b Open the Message Defaults tab and click **Advanced**.
- c In the Message Stream Interface section, select the following options:
  - **Agent MSI**
  - **Divert Messages**

## 3 Assign and distribute the Correlation Composer policy to agents.

- a Locate and highlight the **Correlation Composer** policy.
- b In the context menu, click **Assign to Node / Node Group**.
- c Select the nodes for which you have enabled output to the MSI in [step 1](#).
- d Click **OK** to assign the policy to the selected agent nodes.
- e Click **Deployment**→**Deploy Configuration**.

Alternatively, use the following commands to assign and distribute the Composer EC Policy to the agents:

```
opcnode -assign_pol pol_name='Correlation Composer'
pol_type=ec node_name=<nodename> net_type=NETWORK_IP
opcragt -distrib <nodename>
```

## 4 Verify that Composer and ECS are configured and running.

On the agent, do the following:

- Run **ovc -status** to see these two ECS processes:

opceca	OVO Event Correlation	AGENT, EA	(3238)	Running
opcecaas	ECS Annotate Server	AGENT, EA	(3286)	Running

- Run **ecsmgr -i 12 -info** to verify that the Composer circuit (ecs\_comp) is installed:

```
circuits enabled in stream - circuit ecs_comp
```

After the distribution to the managed nodes completes, ECS, the Composer circuit, and the associated Composer Fact Stores are running on each of the selected managed nodes.

# Configuring MSI in HPOM 9.00 for Windows

After you create correlators in the Composer GUI, and before ECS can begin to process the messages, you need to configure the Message Stream Interface (MSI). ECS can operate on the management server, the managed nodes (agents), or both.

## Configuring MSI on the HPOM 9.00 for Windows Management Server

To configure MSI on the HPOM 9.00 for Windows management server, follow these steps:

### 1 Enable output on MSI.

- a Click **Actions**→**Server**→**Configure**.
- b In the Message Stream Interface section, select the **Enable Output** check box.
- c *Optional:* Send all messages to the Server MSI.

Select one of the following check boxes:

#### — Divert Messages

Default. Divert all messages to the Server MSI.



If you select Divert Messages, all messages are processed by ECS, even if they will never be used. This processing could negatively affect the performance of the server. If you use HP Dependency Mapping Automation (DMA), you must create a `msiconf` file to serialize Composer and the DMA MSI. If you do not create this file, messages that are dropped by Composer continue to appear in the message browser as DMA “tunnels” them around Composer.

#### — Copy Messages

Copy all messages to the Server MSI. All original messages (before ECS processing) are already output, so suppression has no purpose. Enriched messages are provided in addition to, rather than instead of, the original messages.

The management server forwards all messages to the Server MSI. As a result, you do not need to configure specific message types to be forwarded (from the policy configuration).

## 2 Configure specific messages to pass to MSI.

If you chose the Divert Messages option in [step 1](#), all messages are forwarded to the Server MSI, and this step is not required.

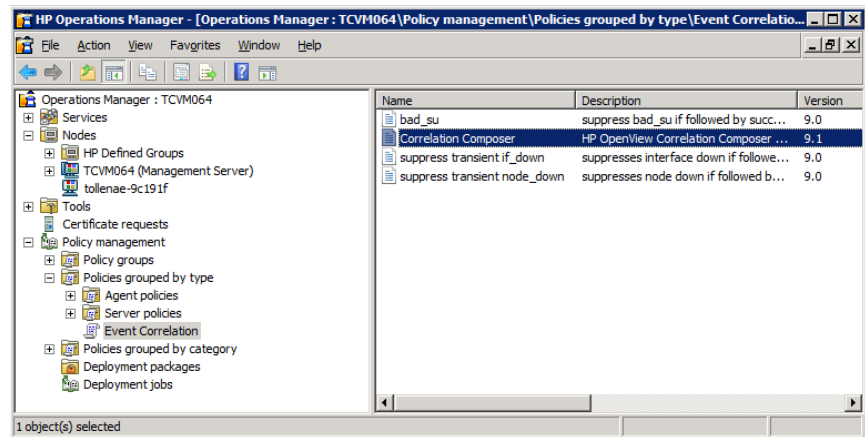
For each policy that you want to pass messages to MSI, follow these steps:

- a In the console tree, go to the Policy management folder.
- b Locate and highlight the policies that produce the messages that must be input to Composer.
- c Click **Modify**→**Advanced Options**.
- d In the Message Stream Interface section, select the following options:
  - **Server MSI**
  - **Divert Messages**

## 3 Assign and distribute the Composer EC Policy to the server.

- a In the left pane, locate the Correlation Composer policy in the following directory:

\Policies\Policies grouped by type\Event Correlation



- b Right-click the **Correlation Composer** policy, select **All Tasks**→**Deploy On**, and then select the management server.
- c Click **OK**.

#### 4 Verify that Composer and ECS are configured and running.

On the management server, do the following:

- a Open the Task Manager to verify that the ECS process (OvOWECM.exe) is running.
- b Run **ecsmgr -i 11 -info** to verify that the Composer circuit (ecs\_comp) is installed:

```
circuits enabled in stream - circuit ecs_comp
```

The MSI is now enabled and the ECS engine is now running.

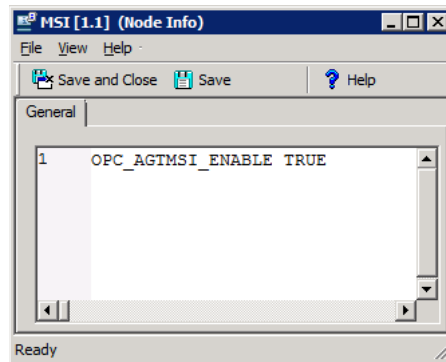
## Configuring MSI on HPOM 9.00 for Windows Managed Nodes (Agents)

To configure MSI on HPOM 9.00 for Windows managed nodes (agents), follow these steps:

#### 1 Enable output on MSI.

- a Create a new Node Info policy with the following content:

```
OPC_AGTMSI_ENABLE TRUE
```

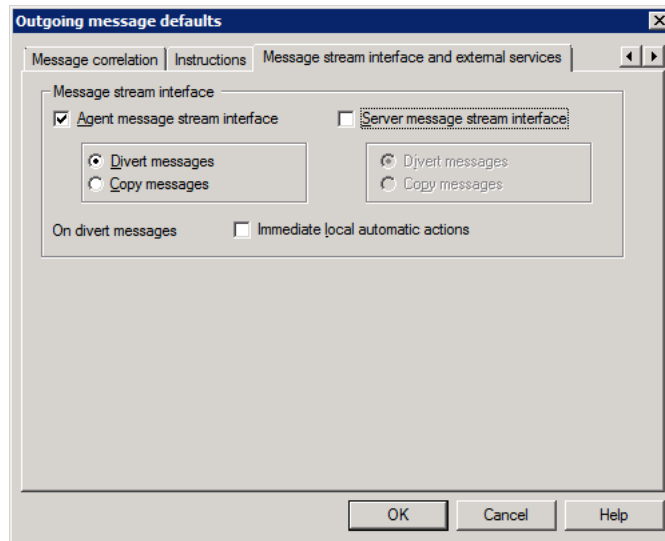


- b Save and close the policy with the title **Enable MSI**.
- c Deploy this policy to the managed nodes where ECS and the Composer circuit (ecs\_comp) are to be run.

## 2 Configure specific messages to pass to MSI.

For each message policy that you want to pass messages to MSI, follow these steps:

- a Go to the Policy Rule screen.
- b Locate and open the rule that produces the messages that must be input to Composer.
- c Click **Modify**→**Actions**→**Message**.
- d Click the **Message Stream Interface and External Services** tab.



- e Select the following:
  - **Agent Message Stream Interface**
  - **Divert Messages**
- f Click **OK**.
- g Update the policy on the agent.



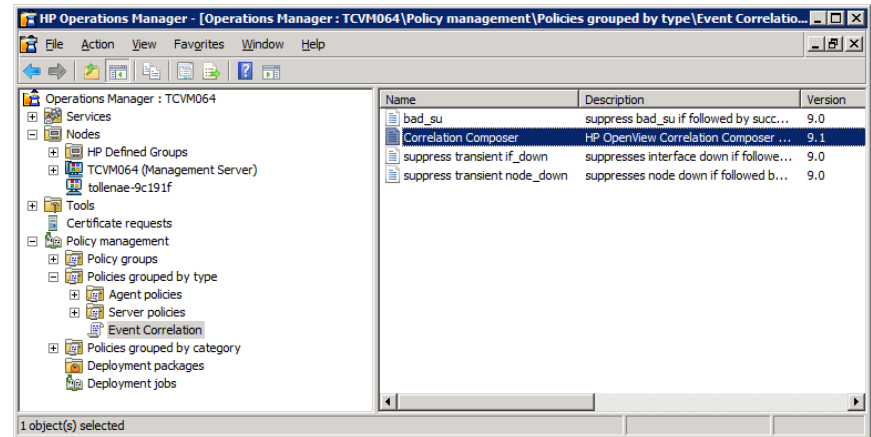
If you want, you can enable MSI for all rules in a policy by going the Rules tab, clicking **Defaults**, clicking the **Message Stream Interface and External Services** tab, and then selecting **Agent Message Stream Interface** and **Divert Messages**.



### 3 Assign and distribute the Composer EC Policy to agents.

- a In the left pane, locate the Correlation Composer policy in the following directory:

\Policies\Policies grouped by type\Event Correlation



- b Right click the Correlation Composer policy, click **All Tasks**→**Deploy On**, and select the nodes you want ECS composer to be active on.

### 4 Verify that Composer and ECS are configured and running.

On the agent, do the following:

- Run **ovc -status** to see these two ECS processes:  

```
opceca OVO Event Correlation AGENT,EA (3238) Running
opcecaas ECS Annotate Server AGENT,EA (3286) Running
```
- Run **ecsmgr -i 12 -info** to verify that the Composer circuit (ecs\_comp) is installed:  

```
circuits enabled in stream - circuit ecs_comp
```

# Merging and Deploying Correlator Store Files

Composer saves all Correlator Stores in files (that is, in ECS Fact Stores). You can use the Composer GUI to create and maintain any number of Correlator Stores. However, the ECS engine can process only one Correlator Store for the Correlation Composer policy. If you have multiple Correlator Stores, the deployment process first merges the files into a single, temporary Correlator Store file (`ecs_comp.fs`).

## Location of Correlator Store Files

The Correlator Store files are located in the following directory:

- *UNIX*

`/etc/opt/OV/share/conf/OpC/mgmt_sv/CO`

- *Windows*

`%OvInstallDir%\conf\OpC\mgmt_sv\CO`

This directory includes a set of sample correlators for HPOM in the `demo.fs` file. To deploy Correlator Stores, use the **ovocomposer -install** command. This command searches the directory for all Correlator Store (`.fs`) files and displays their file names. You can then select the files you want to merge and deploy.

## Composer Applications on UNIX

The HPOM for UNIX Application Bank includes three applications for Composer:

Composer UI

Launches the Composer GUI.

Install Correlators on Agents

Merges and deploys the Composer configuration to selected managed nodes.

Install Correlators on Server

Merges and deploys the Composer configuration to the management server.

## Merging and Deploying on the Management Server

To merge and deploy Composer files on the management server, run the following command:

```
ovocomposer -install [-fs file1 [-fs file2...]] -ms
```

➤ If you do not specify an `-fs` option, the command displays a list of numbered files from that directory, and prompts you for the files to be merged and deployed. In response, you specify a comma-separated list of numbers at the prompt. For details, see the *ovocomposer* reference page.

Example:

```
ovocomposer -install -fs demo.fs -fs myConfig.fs -ms
```

This sample command merges the `demo.fs` and `myConfig.fs` files in the following directory:

- *UNIX*

```
/etc/opt/OV/share/conf/OpC/mgmt_sv/CO
```

- *Windows*

```
%OvInstallDir%\conf\OpC\mgmt_sv\CO
```

The command merges `demo.fs` and `myConfig.fs` into a temporary file (`ecs_comp.fs`), which it then deploys. The deployed file must have exactly the same name as the ECS circuit (`ecs_comp.eco`) that is referenced by the Correlation Composer policy.

➤ In HPOM 9.x for UNIX and Linux, you assign and deploy EC policies just like any other types. However, you can still use the `ovocomposer -install` command to merge multiple correlator files.

## Merging and Deploying on Managed Nodes (Agents)

To merge and deploy Composer files on managed nodes, run the following command:

```
ovocomposer -install [-fs file1 [-fs file2...]] -agt node1  
[node2 ...]
```



If you do not specify an `-fs` options, the command displays a list of numbered files from that directory, and prompts you for the files to be merged and deployed. In response, you specify a comma-separated list of numbers at the prompt. For details, see the *ovocomposer* reference page.

Example:

```
ovocomposer -install -fs demo.fs -fs myConfig.fs -agt  
myAgentNode1.my.com myAgentNode2.my.com
```

This sample command merges the `demo.fs` and `myConfig.fs` files in the following directory:

- *UNIX*  
`/etc/opt/OV/share/conf/OpC/mgmt_sv/CO`
- *Windows*  
`%OvInstallDir%\conf\OpC\mgmt_sv\CO`

The command merges `demo.fs` and `myConfig.fs` into a temporary file (`ecs_comp.fs`), which it then deploys. The deployed file must have exactly the same name as the ECS circuit (`ecs_comp.eco`) that is referenced by the Correlation Composer policy.

# Accessing External Data

You can access extra information for Composer by using the ECS Data Store file and Perl scripting.

## Data Store File

The simplest way to access extra information is by using the ECS Data Store file. The data in this file is normally static. However, you can update it manually, if needed.

### Location of the Data Store File

You must name the Data Store file `ecs_comp.ds` and store it in the following directory:

- *Management server*
  - HPOM for UNIX or Linux  
`/var/opt/OV/shared/server/datafiles/policies/ec`
  - HPOM for Windows  
`%OvDataDir%\shared\server\datafiles\policies\ec`
- *Managed node*
  - HPOM for UNIX  
`/var/opt/OV/conf/eaagt`
  - HPOM for Windows  
`%OvDataDir%\conf\eaagt`



You must manually copy the Data Store to the correct location on each managed node where it is required.

In Composer, you use the `lookup` function to access the ECS Data Store file. For information about the `lookup` function, see [Variables](#) on page 47.



All HPOM message attributes in Composer are of the data type `STRING`, even numeric values. You cannot use numeric values as index values for lookups.

## Syntax of the Data Store File

Each line of the Data Store file uses the following syntax:

```
ADD DATA(keyValue, ReturnValue)
```

This syntax contains the following values:

keyValue

Must be an integer or string.

ReturnValue

Can be any data type.

The Data Store file can contain multiple lines.

The first line in the file must be the header with the following format:

```
#path#date#version#0
```

You begin the comment with two hyphens (--).

Example:

The Data Store is loaded and contains one entry:

```
ADD DATA("Overheated", 80)
```

An X variable has a value of "Overheated". If you use X as a parameter of Lookup, a value of 80 is returned.

Two variables, Y and Z, with values of "Over" and "heated", respectively, result in a key value of "Overheated". The value 80 is returned.



Typically, the Data Store contains static topological information. For example, you could run scripts once a day to create the Data Store file and update the ECS engine with the newly created file.

### Example 1: Creating a New Data Store

To create a new Data Store on the management server, you create an `ecs_comp.ds` file:

```
ecsmgr -i 11 -disable ecs_comp
```

```
ecsmgr -i 11 -circuit_unload ecs_comp
```

```
ecsmgr -i 11 -data_load ecs_comp ecs_comp.ds
```

```
ecsmgr -i 11 -circuit_load ecs_comp ecs_comp.eco ecs_comp ecs_comp
```

```
ecsmgr -i 11 -enable ecs_comp
```



To create a new Data Store on the agent, replace `-i 11` with `-i 12`.

### Example 2: Updating an Existing Data Store

To update an existing Data Store on the management server, you change the date/version string in the `ecs_comp.ds` file:

```
ecsmgr -i 11 -disable ecs_comp
```

```
ecsmgr -i 11 -circuit_unload ecs_comp
```

```
ecsmgr -i 11 -data_update ecs_comp ecs_comp.ds
```

```
ecsmgr -i 11 -circuit_load ecs_comp ecs_comp.eco ecs_comp ecs_comp
```

```
ecsmgr -i 11 -enable ecs_comp
```



To update an existing Data Store on the agent, replace `-i 11` with `-i 12`.

If the Composer Fact Store has already been updated (for example, by using `ovocomposer -install -ms`), you may need to reload it.

## Perl Scripts

By calling external Perl functions, you can access any external data. Typically, Perl functions are used when the data is dynamic in nature.

In the HPOM environment, you must manually copy the Perl module file to the correct location on each managed node where it is required. To find out how to use Perl scripting in Composer, see [Writing Perl Functions](#) on page 113.





## 7 Use Cases in NNM

To help you understand the workflow schema of HP Correlation Composer in the HP Network Node Manager (NNM) environments, this chapter provides the following use cases:

- [Case 1: Enhance Correlation](#) on page 162
- [Case 2: Multi-Source Correlation](#) on page 165
- [Case 3: Rate Correlation](#) on page 170
- [Case 4: Repeated Correlation](#) on page 175
- [Case 5: Suppress Correlation](#) on page 179
- [Case 6: Transient Correlation](#) on page 182
- [Case 7: Multi-Event Correlation](#) on page 188

You can follow these examples when planning your system configuration.



Composer supports the following event types:

- HP Operations Manager (HPOM)
- Simple Network Management Protocol (SNMP)

Although all of the alarm examples in this document use the SNMP Trap Protocol Data Unit (PDU) format, Composer is format-independent.

## Case 1: Enhance Correlation

One use case for Enhance correlation is a Temperature alarm.

### PDU for a Temperature Alarm

A sample SNMP trap PDU for a Temperature alarm could appear in an event log, as follows:

```
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
}
}
}
```

The problem is not immediately evident from the alarm.

The requirement is to add a variable binding with the following string:

```
"Temperature of the device is too high - Please check for
air-conditioning and/or fan failure"
```

## Responding to the Temperature Alarm

When responding to the sample [PDU for a Temperature Alarm](#) on page 162, you would use the Enhance Correlator Template window.

To respond to the Temperature alarm, you would follow these steps:

- 1 **Identify the Temperature alarm.**

All Temperature alert failures have the following identifying attributes:

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

- 2 **Differentiate Temperature ON and OFF alarms.**

If specific-trap is set to 95, the alarm is a Temperature ON alarm.

- 3 **Respond to the original alarm.**

You can choose to retain the original alarm along with the enhanced alarm. [Table 6](#) describes the functionality of the buttons in the Enhance Correlator Template window.

**Table 6 Buttons in the Enhance Correlator Template Window**

Button Name	Selected	Functionality
Want Original	Yes	Original alarm is output with the enhanced alarm.
	No	Only the enhanced alarm is output. The original alarm is discarded.
Enhance Always	Yes	Alarms are modified and output, even if another correlation discards the alarm.
	No	Alarms are modified if no other correlator discards the alarm.



Click the **Enhance Always** button with caution. By clicking this button, you enhance all alarms.

## Defining the Enhance Correlator Template

When responding to the [PDU for a Temperature Alarm](#) on page 162, you would define an Enhance correlator template.

To define the Enhance correlator template, follow these steps:

- 1 From the Correlator Store window, click **Correlations**→**Correlator Templates**→**Enhance**.

The Enhance Correlator Template window opens.

- 2 In the Name text box, type a name for the correlator.
- 3 In the Description text box, type a description for the correlator.
- 4 To define the Alarm Signature, type the following values:

- **enterprise = 1.2.3.4.995**
- **generic-trap = 6**
- **specific-trap = 95**

- 5 Declare the `errstr` variable:

- a In the Name cell, type **errstr**.
- b From the Operator drop-down menu, select **Constant**.
- c In the Value field, type the following string:

**"Temperature of the device is too high - Please check for air-conditioning and/or fan failure"**

These substeps are the equivalent of `errstr` constant `<str>`.

- 6 Click the **New Alarms** tab to alter the alarm.

The New Alarm panel opens.

- 7 From the drop-down menu, select **Alter Specification**.

The Alter Alarm Definition table displays.

- 8 To alter the alarm, define the following attributes:

- From the Field drop-down menu, select **variable-bindings[1].value**.
- From the Mode drop-down menu, select **replace**.
- From the Value drop-down menu, select **errstr**.

- 9 Click **OK** to complete the definition of the correlator.

The new correlator displays in the Correlator Store table.

## Case 2: Multi-Source Correlation

One use case for Multi-Source correlation is multiple redundant Signaling System Number 7 (SS7) links between two switching entities. Together, these SS7 links form a logical entity, called a signaling set. If the trunk between the two links fails, an SS7 link set failure is received, with an 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.

### PDU for SS7 Link Failure

An SNMP trap PDU for an SS7 link failure could appear in an event log, as follows:

```
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"
}
}
}
```

### PDU for SS7 Link Set Failure

An SNMP trap PDU for an SS7 link set failure could appear in an event log, as follows:

```
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"
}
}
}
```

## Responding to the SNMP Trap PDU Alarms

When responding to the sample [PDU for SS7 Link Failure](#) on page 165 and the sample [PDU for SS7 Link Set Failure](#) on page 165, you would use the Multi-Source Correlator Template window.

To respond to the SS7 link and link set failures, you would follow these steps:

- 1 Identify the SS7 link failure alarms.**

All SS7 link failure alarms have the following identifying attributes:

- enterprise is set to 1.2.3.4.997.
- generic-trap is set to 6.
- specific-trap is set to 55.

- 2 Identify the SS7 link set failure alarms.**

All SS7 link set failure alarms have the following identifying attributes:

- enterprise is set to 1.2.3.4.999.
- generic-trap is set to 6.
- specific-trap is set to 56.

- 3 Determine whether the SS7 link and SS7 link set failures are emitted from the same device and belong to the same set.**

The SS7 link failure belongs to the SS7 link set failure only if both of the following are true:

- SS7 link set ID and the SS7 link ID are the same.
- Agent addresses for both the failure alarms are the same.

#### 4 Respond to the alarms.

You can discard alarms, based on whether the set is complete. [Table 7](#) describes the functionality of the various buttons in the Multi-Source Correlator Template window.

**Table 7 Buttons in the Multi-Source Correlator Template Window**

Button Name	Selected	Functionality
Discard on Set Completion	Yes	If the set is complete, the alarm is discarded. Otherwise, the alarm forwarded.
	No	Alarm is forwarded, regardless of set completion.
Window Period		Mandatory field. Time period in 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. For details, see <a href="#">Multi-Source Correlator Template</a> on page 32.
	Yes	Operates in Mode 2. For details, see <a href="#">Multi-Source Correlator Template</a> on page 32.

## Defining the Multi-Source Correlator Template

When responding to the sample [PDU for SS7 Link Failure](#) on page 165 and the sample [PDU for SS7 Link Set Failure](#) on page 165, you would define a Multi-Source correlator template.

To define the Multi-Source correlator template, follow these steps:

- 1 From the Correlator Store window, click **Correlations**→**Correlation Templates**→**Multi-Source**.

The Multi-Source Correlator Template window opens.

- 2 In the Name text box, type a name for the correlator.
- 3 In the Description text box, type a description for the correlator.
- 4 Click the **Definition** tab to open the Alarm Definition panel.
- 5 In the Name panel on the left side of the window, type a name for the SS7 link failure alarm.
- 6 Define the Alarm Signature to identify the SS7 link failure alarm.

In the Alarm Signature table, type the following values:

- **enterprise id=1.2.3.4.997**
- **generic trap=6**
- **specific trap=55**

- 7 In the Variables table, declare the following variables:

- *Name of the SS7 link ID variable*

The SS7 link ID is extracted from `variable-bindings[0].value`.

- *Extract pattern*

In the Extract Pattern window, type **\*#-#<#.linkID>\***.

- *Pattern separator*

Set to **"#"**.

This step extracts the SS7 link ID and assigns the extracted numeral to the variable `linkID`.



- 8 Define the message key:
  - a Click the **Message Key** text box.
  - b In the shortcut menu, click **SS7 Link Failure**→**SS7 Link ID**→**linkID**.
- 9 To create a set of alarms, select the **Set** check box.
- 10 *Optional:* To alter the alarm, define the changes in the Alter Alarm Definition table.
- 11 *Optional:* To add a new alarm, right-click and select **Add**.  
Do the following:
  - 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**.
- 12 In the Parameters panel, type the following parameters:
  - 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 is not emitted out until a Power ON alert is received.
- 13 Click **OK** to complete the definition of the Signature file.  
The new correlation displays in the Correlator Store table.

## Case 3: Rate Correlation

One use case for Rate correlation is radio antenna failure. Radio antennas frequently report failure during bad weather conditions. The requirement is to discard all radio antenna failures if the rate of failure is below five failures in 30 minutes. If the rate exceeds this threshold, you forward the alarm to the browser after annotating the alarm with the rate.

### PDU for Radio Antenna Failure

An SNMP trap PDU for an antenna failure could appear in an event log, as follows:

```
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"
}
}
}
```

## Responding to Radio Antenna Failure Alarms

When responding to the sample [PDU for Radio Antenna Failure](#) on page 170, you would use the Rate Correlator Template window.

To respond to the radio antenna failure alarm, you would follow these steps:

### 1 Identify the alarms for which the count is maintained.

A count is maintained for alarms identified by the following attributes:

- enterprise is set to 1.2.3.4.998.
- generic-trap is set to 6.
- specific-trap is set to 80.
- variable-bindings[0].value is set to 2.

### 2 Respond to the alarms.

You can discard duplicate alarms. However, you must define the correlation to monitor the time at which the alarm is discarded or output. [Table 8](#) describes the functionality of the buttons in the Rate Correlator Template window.

**Table 8 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. Threshold count. If the number of alarms exceeds the threshold count within the specified window period, the rate threshold is considered to be breached.
Discard	Yes	All alarms are discarded. If a new alarm is created, it is output. No other alarms are output.
	No	Alarms are <i>not</i> discarded. If a new alarm is created, it is output.

## Defining the Rate Correlator Template

When responding to the sample [PDU for Radio Antenna Failure](#) on page 170, you would define a Rate correlator template.

To define the Rate correlator template, follow these steps:

- 1 From the Correlator Store, click **Correlations**→**Correlator Templates**→**Rate**.

The Rate Correlator Template window opens.

- 2 In the Name text box, type a name for the correlator.
- 3 In the Description text box, type a description for the correlator.
- 4 Click the **Definition** tab to display the Alarm Definition panel.
- 5 To identify the Alarm Signature, type the following values:

- **enterprise**
- **generic-trap**
- **specific-trap**

- 6 Declare the following variables:

- `ant`

Together with the extracted pattern, this variable specifies the antenna ID and base transceiver station (BTS) ID:

- Extract the antenna ID and BTS from `variable-bindings[1].value`.

In the extract pattern window, type the following:

**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` variables are the same.

- `mkey`

Unique field that combines all of the attributes into one attribute, which constitutes the message key. Combine the attributes `ant.antid`, `ant.btsid`, and `agent-addr`.

- 7 Select the message key.
  - a Click the Message Key window.

A pop-up menu displays all attributes and predefined variables.
  - b From the pop-up menu, click **mkey**.
- 8 Define the parameters for the correlation:
  - **Window Period = 30 minutes**
  - **Count = 5**
- 9 *Optional:* If you want to discard the alarms, click **Discard**.

After the alarms are discarded, the count of alarm arrival is still maintained.
- 10 In the Variables table, define the following variables:
  - **str1 constant "The threshold has been breached for the antenna "**
  - **str2 constant "from BTS"**
  - **errstr combine of str1, ant.antid, str2, ant.btsid**

This definition creates an `errstr` that looks like the following:

```
"The threshold has been breached for the antenna 10 from BTS
20"
```

Before a new alarm is created, it is necessary to define the error string that declares the problem.
- 11 Define the new alarm.

To alter the alarm, click the **New Alarms** tab.

The New Alarm panel opens.
- 12 From the drop-down menu, select **New Alarm Specification**.

The New Alarm Definition table displays.

- 13 To define the change, select the following:
- `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 **OK** to complete the definition of the correlator.
- The correlator is displayed in the Correlator Store table.

## Case 4: Repeated Correlation

One use case for Repeated correlation is duplicate alarms. In general, 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. This correlation monitors duplicate alarms arriving within the specified window period.

When utilization exceeds the threshold, routers generate a CPU Hog alarm. 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. At the end of the 30-minute period, a new alarm is generated. The new alarm indicates the number of such alarms received (and discarded).

### PDU for Duplicate Alarms

An SNMP trap PDU for duplicate alarms could appear in a log, as follows:

```
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 { }
}
```

## Responding to Duplicate Alarms

When responding to the sample [PDU for Duplicate Alarms](#) on page 175, you would use the Repeated Correlator Template window.

To respond to duplicate alarms, you would follow these steps:

- 1 Identify which alarms are duplicated.**

Duplicate alarms have the following attributes:

- enterprise is set to 1.2.3.4.6.
- generic-trap is set to 6.
- specific-trap is set to 25.

- 2 Identify which alarms are emitted from the same router.**

Two alarms are said to come from the same router if the agent address of the router from which they are emitted is the same.

- 3 Respond to the duplicate alarms.**

You can respond to duplicate alarms by placing them in one of the following states:

- **Discarded**

Events are discarded from HP Event Correlation Services (ECS). After that, the events are not available for further correlation.

- **Output**

Before outputting events, you need to decide whether they should take part in other correlations.



#### 4 Set a time to discard or output the alarms.

You can choose to discard or output alarms whenever required, based on the correlator definition. [Table 9](#) describes the functionality of the buttons in the Repeated Correlator Template window.

**Table 9 Buttons in the Repeated Correlator Template Window**

Button	Selected	Functionality
Window Period	N/A	Mandatory field. Time period for which the alarm duplication is monitored.
Discard Duplicate	Yes	Chooses Mode 1 of operation. For details, see <a href="#">Repeated Correlator Template</a> on page 35.
	No	Chooses Mode 2 of operation. For details, see <a href="#">Repeated Correlator Template</a> on page 35.
Discard Immediately	Yes	Applicable only if you select the Discard Duplicate button. All duplicate alarms are discarded without participating in other correlations.
	No	Duplicate alarms are discarded only after participating in other correlators.

## Defining the Repeated Correlator Template

When responding to the sample [PDU for Duplicate Alarms](#) on page 175, you would define a Repeated correlator template.

To define the Repeated correlator template, follow these steps:

- 1 From the Correlator Store window, select **Correlations**→**Correlator Templates**→**Repeated**.

The Repeated Correlator Template window opens.

- 2 In the Name text box, type a name for the correlator.
- 3 In the Description text box, type a description for the correlator.
- 4 Click the **Definition** tab to display the Alarm Definition panel.

- 5 To define the Alarm Signature, type the following values:
  - **enterprise = 1.2.3.4.6**
  - **generic-trap = 6**
  - **specific-trap = 25**
- 6 Declare the following variables:
  - **str constant "alarms discarded in 30 minutes"**
  - **errstr combine of AlarmCnt and str**
- 7 Select the following:  
**Message Key = agent-addr**
- 8 Define the window period for which the alarm arrival must be maintained.
- 9 In the Window Period field, type **30 minutes**.
- 10 Select the following buttons:
  - **Discard Duplicate**
  - **Discard Immediately**

Discard duplicate alarms as soon as they have taken part in this correlation. You do not want them to take part in other correlations.
- 11 Click the **New Alarms** tab.  
The New Alarm panel displays.
- 12 From the drop-down menu, select **New Alarm Specification**.  
The Create Alarm Definition table displays.
- 13 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**
- 14 Click **OK** to complete the definition of the correlator.  
The correlator displays in the Correlator Store table.

## Case 5: Suppress Correlation

One use case for Suppress correlation is movement alarms. Normally, movement traps require investigation. However, if the movement alarms are from exchanges emitted from the City offices, they can be discarded because 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.

### PDU for Movement Alarms

An SNMP trap PDU for movement alarms could appear in a log, as follows:

```
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"
}
}
}
```

## Responding to Movement Alarms

When responding to the sample [PDU for Movement Alarms](#) on page 179, you would use the Suppress Correlator Template window.

To respond to movement alarms, you would follow these steps:

### 1 Identify which alarms to suppress.

All movements are identified by the following attributes:

- `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` contains the string "City".
- `variable bindings[2].value` contains the following string:  
"There is Movement"

### 2 Identify the parameters to configure.

[Table 10](#) describes the functionality of the different buttons in the Suppress Correlator Template window.

**Table 10 Buttons in the Suppress Correlator Template Window**

Button Name	Selected	Functionality
Participate In Other Correlation	No	Alarm does <i>not</i> participate in other correlations before it is discarded.
	Yes	Alarm takes part in other correlations before it is discarded.

## Defining the Suppress Correlator Template

When responding to the sample [PDU for Movement Alarms](#) on page 179, you would define a Suppress correlator template.

To define the Suppress correlator template, follow these steps:

- 1 From the Correlator Store window, select **Correlations**→**Correlator Templates**→**Suppress**.

The Suppress Correlator Template window opens.

- 2 In the Name text box, type a name for the correlator.
- 3 In the Description text box, type a description for the correlator.
- 4 Click the **Definition** tab to display the Alarm Definition panel.
- 5 To define the Alarm Signature, set the following values:
  - **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"**
- 6 Click **OK** to complete the correlator.

The correlator displays in the Correlator Store table.

## Case 6: Transient Correlation

One use case for Transient correlation is pulse code modulation (PCM) link failure. PCM links get out of synchronization frequently. When they do, a trap is generated that indicates a link failure. Typically, the two ends of the PCM link re-synchronize within one second. The requirement is to hold the PCM Down alarm for a period of two seconds, and to discard it if the PCM Up alarm is received within this time period. Also, if five such link failures are detected in a 30-minute period, a new alarm must be generated to indicate instability. The newly created alarm indicates the time taken for the breach to take place.

### PDU for PCM Link Failure

A SNMP trap PDU for PCM link failure would appear in a log, as follows:

```
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
    }
  }
}
```

### PDU for PCM Clear Alarm

A SNMP trap PDU for PCM clear alarm would appear in a log, as follows:

```
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
    }
  }
}
```

## Responding to PCM Link Failure

When responding to the sample [PDU for PCM Link Failure](#) on page 182, you would use the Transient Correlator Template window.

To respond to PCM link failure alarms, you would do the following:

- 1 Identify PCM trap alarms.**

All PCM link alarms are identified by the following attributes:

- enterprise is set to 1.2.3.4.999.
- generic-trap is set to 6.
- specific-trap is contained in [400, 401].

- 2 Differentiate PCM clear and link failure alarms.**

The alarms are distinguished by the specific-trap attribute:

- If specific-trap is set to 401, the alarm is a PCM clear alarm.
- If specific-trap is set to 400, the alarm is a PCM link failure alarm.

- 3 Determine whether the alarms come from the same PCM link.**

Alarms come from the same PCM link if either of the following is true:

- variable-bindings[0].value contains the PCM link ID.
- Two alarms have the same agent-addr and PCM Link ID.

- 4 Determine how time is maintained.**

Determine the following:

- Time interval for correlation to be monitored
- Time period for which a failure alarm waits for a clear alarm to arrive

Table 11 describes the functionality of the various buttons in the Transient Correlator Template window.

**Table 11 Buttons in the Transient Correlator Template Window**

Button Name	Selected	Functionality
Window Period	N/A	Mandatory field. Maximum time a failure alarm is held for a clear alarm. If the clear alarm is received while the alarm is held, 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 and 10 minutes, depending on the severity of the problem.
Enable Threshold	Yes	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 disabled.
Threshold Count	N/A	Number of failure and clear alarm pairs arriving. This button is enabled only if the Enable Threshold button is enabled.
Threshold Window	N/A	Time period for which the count is maintained. This button is enabled only if the Enable Threshold button is enabled.



## Defining the Transient Correlator Template

When responding to the sample [PDU for PCM Link Failure](#) on page 182, you would define a Transient correlator template.

To define the Transient correlator template, follow these steps:

- 1 From the Correlator Store window, select **Correlations→Correlator Templates→Transient**.

The Transient Correlator Template window opens.

- 2 In the Name text box, type a name for the correlator.
- 3 In the Description text box, type a description for the correlator.
- 4 To identify the Alarm Signature, type the following values:

- **enterprise = 1.2.3.4.999**
- **generic-trap = 6**
- **specific-trap is in list [400, 401]**



You must define the Alarm Signature so that the clear and failure alarms pass this condition.

- 5 Declare the following variable:

**clear constant 401**

This variable differentiates the clear alarm from the failure alarm.

- 6 Define the time period within which a clear alarm must arrive after the failure alarm has arrived.

In the Window Period field, enter **2 seconds**.

- 7 Define the clear alarm:

- a Click the **Clear Alarm** button.

The Clear Alarm window opens.

- b From the drop-up menus under the respective headings, select the following:

- **Attribute = Specific-trap**
- **Operator = '='**
- **Value = clear**

8 Define the message key.

The message key is a combination of the PCM link ID and the agent address. The message key uniquely identifies the PCM link from which the alarms are emitted. You must define a variable that contains this value.

To define the variable, follow these steps:

a In the Variables table, enter **PCMLink** in the Name cell.

b Select **Operator = Combine**.

The Combine Definition window opens.

c From the pop-up menu, select the following parameters:

— **agent-addr**

— **variable-bindings[0]->value**

d Close the Combine Definition window.

9 From the message key pop-up menu, select **PCMLink**.

10 Create variables containing the error string:

```
errstr1 constant "Threshold breached in "
```

```
errstr2 constant "seconds"
```

```
errstr = combine of str1, CorrelationDuration, str2
```

The `CorrelationDuration` variable is generated automatically by Composer to monitor the time taken for the threshold to be breached.

11 Define the threshold window period:

a Select the **Enable Threshold** checkbox.

b Set the following values:

```
Threshold Count = 5
```

```
Threshold Window = 30 minutes
```

- 12 Define the new alarm to be output:
  - a Click the **New Alarm** tab.  
The New Alarm panel displays.
  - b From the drop-down menu, select **New Alarm Definition**.  
The New Alarm Definition table displays.  
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 **OK** to complete the definition of the correlator.  
The correlation displays in the Correlator Store table.

## Case 7: Multi-Event Correlation

One use case for Multi-Event correlation is Mobile Switching Center (MSC) and Base Station Controller (BSC) failure. In a network, if an MSC fails, and a connected BSC fails within five minutes, you add a valid message to the BSC indicating what the problem may be.

### PDU for MSC Failure

A sample SNMP trap PDU for an MSC failure could appear in an event log, as follows:

```
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"
}
}
}
```

### PDU for BSC Failure

A sample SNMP trap PDU for a BSC failure could appear in an event log, as follows:

```
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"
}
}
}
```

## Responding to Connected MSC and BSC Failure

When responding to the sample [PDU for MSC Failure](#) on page 188 and the sample [PDU for BSC Failure](#) on page 188, you would use the Multi-Source Correlator Template window.

To respond to a connected MSC and BSC failure in a network, you would follow these steps:

### 1 Identify the MSC failure alarm.

All MSC failures are identified by the following attributes:

- enterprise is set to 1.2.3.4.996.
- generic-trap is set to 6.
- specific-trap is set to 65.

### 2 Identify the BSC failure alarm.

All BSC failures are identified by the following attributes:

- enterprise is set to 1.2.3.4.995.
- generic-trap is set to 6.
- specific-trap is set to 66.

### 3 Determine whether the MSC and BSC are connected.

Studying the alarms alone does not help identify how the alarms are related to each other. To understand the topology, you need to invoke an external application. To extract the required information, you write a user-defined function.

For example, you could define a `getname()` function to take the BSC ID as a parameter and return the name of the MSC to which it is connected. If the value returned is the same as the MSC that emitted the failure alarm, the two devices are connected.

## Defining the Multi-Source Correlator Template

When responding to the sample [PDU for MSC Failure](#) on page 188 and the sample [PDU for BSC Failure](#) on page 188, you would define the Multi-Source correlator template.

To define the Multi-Source correlator template, follow these steps:

- 1 From the Correlator Store window, select **Correlations**→**Correlator Templates**→**Multi-Source**.

The Multi-Source Correlator Template window opens.

- 2 In the Name text box, type a name for the correlator.
- 3 In the Description text box, type a description for the correlator.
- 4 Click the **Definition** tab to display the Alarm Definition panel.
- 5 In the Name panel on the left side of the window, enter the name for the alarm:

**MSC Failure**

- 6 Declare the Alarm Signature to identify the MSC Failure alarms.

In the Definition table, type the following values:

- **enterprise = 1.2.3.4.996**
- **generic-trap = 6**
- **specific-trap = 65**

- 7 Define the `MSCLinkID` variable:

- Type the name of the variable:

**MSC Link ID**

MSC link ID is extracted from `variable-bindings[0].value`.

- Type the set to extract:  
**\*#-#<#.mscID>\***
- Set the pattern separator set to **"#"**.

This step extracts the MSC link ID and binds it to the variable `mscID`.

- 8 Define the message key:
  - a Click the **Message Key** text box.  
A pop-up menu displays.
  - b Select **MSC Failure**→**MSCLinkID**→**mscID**.
- 9 *Optional:* If you want to alter the alarm, define the changes in the Alter Alarm Definition table.
- 10 Add a new alarm:
  - a Right-click the mouse button in the Name panel on the left.
  - b Enter **BSC Failure**.
  - c Define the Alarm Signature to identify BSC Failure:
    - **enterprise = 1.2.3.4.995**
    - **generic-trap = 6**
    - **specific-trap = 66**
  - d Declare a variable BSCLinkID extracted from variable-bindings[0].value.
  - e Select the message key to **BSC Failure**→**MSCName**.
- 11 To identify how the MSC and BSC are connected, define a function that performs the required operation.  
  
For example, you can define a `getname()` function that takes the BSC name as a parameter and returns the name of the MSC to which it is connected.
- 12 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**
- 13 In the Parameters panel, do the following:
  - 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.

- 14 Define the new alarm:
  - a Click the **New Alarm** tab.  
The New Alarm panel displays.
  - b From the drop-down menu, select **New Alarm Definition**.  
The New Alarm Definition displays.
  - c 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**
- 15 Click **OK** to complete the definition of the correlator.  
The correlator displays in the Correlator Store table.



## 8 Use Cases in HPOM

To help you understand the workflow schema of HP Correlation Composer in the HP Operations Management (HPOM) environment, this chapter provides the following use cases:

- [Case 1: Enhance Correlation](#) on page 194
- [Case 2: Suppress Correlation](#) on page 210
- [Case 3: Multi-Source Correlation](#) on page 215
- [Case 4: Rate Correlation](#) on page 225
- [Case 5: Transient Correlation](#) on page 233

You can follow these examples when planning your system configuration.



Although all of the alarm examples in this document use the HPOM message format, Composer is format-independent.

Composer supports the following event type:

- HP Operations Manager (HPOM) Message

Most likely, you will never face a scenario in which Repeated correlation is required in a HPOM environment. The sophisticated facilities available with the standard Message Source template configurations in HPOM are more appropriate in nearly all situations. Should a situation arise where Repeated correlation is required, see [Case 4: Repeated Correlation](#) on page 175 of [Chapter 7, Use Cases in NNM](#).

## Case 1: Enhance Correlation

In the HPOM environment, Enhance correlation is frequently used to enrich messages.

Common examples of message enrichment include the following:

- [Changing Simple Message Text](#) on page 194
- [Replacing Error and Status Codes with Descriptions](#) on page 200
- [Enriching Messages by Using Perl Commands](#) on page 205

### Changing Simple Message Text

As an example of Enhance correlation, you might verify that the OBJECT message attribute is set to Test. If it is, you could add Test: to the beginning of MSGTEXT.

Consider the following two messages:

```
NODENAME:      system1
OBJECT:         Test
APPLICATION:    Web Monitor
MSGTEXT:        Application Memory Warning
```

```
NODENAME:      system1
OBJECT:         disk1
APPLICATION:    Disk Monitor
MSGTEXT:        Volume Warning
```

You need to match and modify the first message only.

After the text enhancement, the message text would read as follows:

```
MSGTEXT:        Test: Application Memory Warning
```

In this scenario, you normally do not output the original message with the enhanced message. That is, you do not present two messages for one condition.

## Changing the Text of a Simple Message

To change the text of a simple message, you would follow these steps:

- 1 **Identify the test message.**

All test messages are identified by the following attribute:

OBJECT is set to "Test".

- 2 **Change the original message.**

You can retain the original alarm along with the enhanced alarm.

[Table 12](#) describes the functionality of the buttons in the Enhance Correlator Template window.

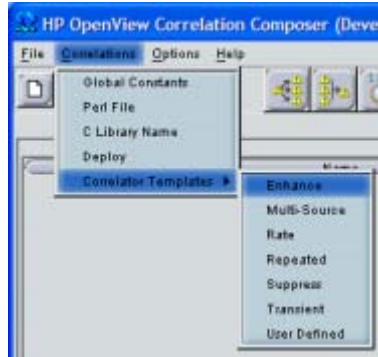
**Table 12 Buttons in the Enhance Correlator Template Window**

Button Name	Selected	Functionality	When to Use
Want Original	Yes	Original message is output with the enhanced message.	Rarely used. Typically, the option is required when the enhanced message is set to represent a separate condition from that of the original. It is still appropriate to show the original message.
	No	Only the enhanced message is output. The original message is discarded.	Default option. If the enhanced message adds enriched information to the original condition, it is not appropriate to show the original message, which duplicates part of the enhanced message.
Enhance Always	Yes	Message is modified and output, even if another correlation discards the message.	Rarely used. Typically, the option is used if the enhancement includes external Perl functions with side effects that should be executed even if the message will be discarded by another correlation.
	No	Message is modified if no other correlator discards it.	Default option. If another correlator suppresses a message that is also to be enriched, the message is not enriched or output.

## Changing Message Text in the Enhance Correlator Template

To change message text in the Enhance correlator template, you would follow these steps:

- 1 Do one of the following:
  - From the Correlator Store window, click **Correlations**→**Correlator Templates**→**Enhance**.



- Click the **Enhance** button.



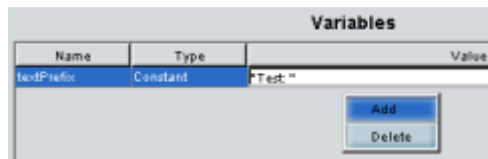
The Enhance Correlator Template window opens.

- 2 In the Name text box, type a name for the correlator.
  - ⚠ You cannot use spaces in the correlator name.
- 3 In the Description text box, type a description for the correlator.

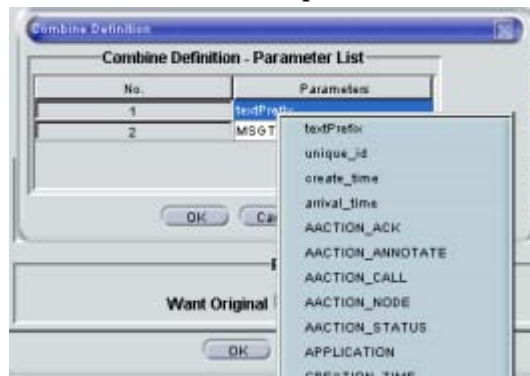


- 4 Click the **Definition** tab.

- 5 To define the Alarm Signature (to select only messages of interest to you), select the following values:
  - a In the Field cell, select **OBJECT**.
  - b From the Operator drop-down menu, select the equal sign (=).
  - c In the Value field, type "**Test**".
- 6 Create a textPrefix variable for the prefix of the message text:
  - a In the Name text box, type **textPrefix**.
  - b From the Operator drop-down menu, select **Constant**.
  - c In the Value field, type the string "**Test**".
- 7 Create a newText variable to concatenate (combine) the prefix text (textPrefix) with the message text attribute:
  - a Press **Enter**, or right-click the **Variables** area and click **Add** in the shortcut menu.



- b In the Name cell, type **newText**.
- c From the Operator field drop-down menu, select **Combine**.
- d Click the **Value** field to open the Combine Definition box.



- e For the first parameter, select **textPrefix**.
- f For the second parameter, select **MSGTEXT**.

The Definition tab should now look like this.

The screenshot shows the 'Enhance' dialog box with the 'Definition' tab selected. The 'Name' field contains 'SimpleMessageTextChange'. Below the tabs are four sections: 'Alarm Signature', 'Variables', 'Advanced Filter', and 'Parameters'.

**Alarm Signature**

Field	Operator	Value
OBJECT	=	"Test"

**Variables**

Name	Type	Value
testPrefix	Constant	"Test:"
newText	Combine	testPrefix,MSGTEXT

**Advanced Filter**

Name	Operator	Value
------	----------	-------

**Parameters**

Want Original ☐ Enhance Always ☐

Buttons: OK, Cancel, Help

- 8 Alter the message:
  - a Click the **New Alarm** tab.

The New Alarm panel opens.
  - b From the drop-down menu, select **Alter Specification**.

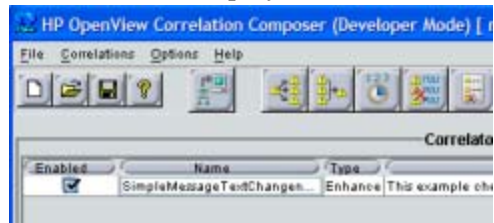
The Alter Alarm Definition table displays.
  - c Define the following attributes:
    - From the Field drop-down menu, select **MSGTXT**.
    - From the Mode drop-down menu, select **Replace**.
    - From the Value drop-down menu, select **newText**.

The New Alarm tab should now look like this.



- 9 Click **OK** to complete the definition of the correlator.

The correlator displays in the Correlator Store table.



## Replacing Error and Status Codes with Descriptions

As an example of Enhance correlation, you might want to verify that the OBJECT message attribute is set to HTTP, and then find the error or status code in the MSGTEXT attribute. If the error code is in the 500 through 599 range, it is a server error. You should replace it with a status description that is retrieved from the Data Store. (In addition, you would use the Suppress correlator to suppress all non-server error or status messages. For details, see [Case 2: Suppress Correlation](#) on page 210.)

Consider the following message and sample Data Store entries:

```
NODENAME:      system1
OBJECT:         HTTP
APPLICATION:    Web Monitor
MSGTEXT:        Response Code: 502

#/tmp/c.ds#Sat Aug 16 15:39:47 2008#0#0
ADD DATA("HTTP_500", "Internal Server Error")
ADD DATA("HTTP_501", "Not Implemented")
ADD DATA("HTTP_502", "Bad Gateway")
ADD DATA("HTTP_503", "Service Unavailable")
ADD DATA("HTTP_504", "Gateway Timeout")
ADD DATA("HTTP_505", "HTTP Version Not Supported")
ADD DATA("HTTP_506", "Variant Also Negotiates")
ADD DATA("HTTP_507", "Insufficient Storage")
ADD DATA("HTTP_509", "Bandwidth Limit Exceeded")
ADD DATA("HTTP_510", "Not Extended")
```

For this message, you would replace the MSGTEXT value with Bad Gateway:

```
MSGTEXT:        Bad Gateway
```



## Replacing an Error or Status Code with a Description

To replace an error or status code with a description, you would follow these steps:

- 1 **Identify appropriate messages.**

All HTTP messages are identified by the following attributes:

- OBJECT is set to "HTTP".
- MSGTEXT has a format such as the following:

"Response Code: 502"

- 2 **Decide what to do with the original message.**

You output the modified message and suppress the matching messages. For a detailed description of options, see [Table 12](#) on page 195.

## Replacing a Code with a Description in the Enhance Correlator Template

To replace an error or status code with a description in the Enhance correlator template, follow these steps:

- 1 Do one of the following:

- From the Correlator Store window, click **Correlations**→**Correlator Templates**→**Enhance**.



- Click the **Enhance** button.



The Enhance Correlator Template window opens.

- 2 In the Name text box, type a name for the correlator.



You cannot use spaces in the correlator name.

- 3 In the Description text box, type a description for the correlator.
- 4 Click the **Definition** tab.
- 5 To define the Alarm Signature (to select only messages of interest to you), select the following values:
  - a In the Field cell, select **OBJECT**.
  - b From the Operator drop-down menu, select the equal sign (=).
  - c In the Value field, type **"HTTP"**.
- 6 Create a `errorCode` variable to hold the error or status code to be extracted with a pattern from the matching text:
  - a In the Name text box, type **errorCode**.
  - b From the Operator drop-down menu, select **Extract**.




Pattern matching in Composer uses the syntax of the HPOM template configuration.

- c In the Attribute pop-up menu, select **MSGTEXT**.
- d In the Pattern text box, type **Response Code: <\*.code>\$**.

The pattern separator is not required to specify whether normal whitespace is used as the word separator. In this case, you can later use the `code` tag to represent that part of the extracted message text (the HTTP status or error code).



- 7 Create a `lookupPrefix` variable to hold the prefix to the extract code:
  - a In the Name cell, type **lookupPrefix**.
  - b From the Type field drop-down menu, select **Constant**.
  - c In the Value field, enter **"HTTP\_"**.
- 8 Create a `lookupCode` variable to hold the text that is combined with the `lookupPrefix` and the extract code to look up the Data Store:
  - a In the Name cell, type **lookupCode**.
  - b From the Type field drop-down menu, select **Combine**.
  - c In the Value field, select the following:
    - **lookupPrefix**
    - **errorCode->code**
- 9 Create a message variable to hold the text set from the Data Store lookup:
  - a In the Name cell, type **message**.
  - b From the Type field drop-down menu, select **Lookup**.
  - c In the Value field, select **lookupCode**.

 The Lookup operator requires that the data to be retrieved is already loaded in the Data Store. For details, see [Data Store File](#) on page 157.
- 10 Create a `desiredRange` variable to hold the pattern used in the Advanced Filter to select only messages that are for HTTP code or status values in the range 500 through 599:
  - a In the Name cell, type **desiredRange**.
  - b From the Type field drop-down menu, select **Constant**.
  - c In the Value field, type **"5<2#>"**.

You use the Advanced Filter to select and enhance a subset of messages that match the Alarm Signature.

- 11 Declare the Advanced Filter entry, `errorCode.code`, to match the pattern declared in `desiredRange`:
  - a In the Name cell, select **errorCode->code**.
  - b From the Operator field drop-down menu, select **matches**.
  - c In the Value field, select **desiredRange**.

The Definition tab should now look like this.

The screenshot shows the 'Enhance' dialog box with the 'Definition' tab selected. The 'Name' field contains 'httpServerStatus'. Below the tabs are three sections: 'Alarm Signature', 'Variables', and 'Advanced Filter'.

**Alarm Signature**

Field	Operator	Value
OBJECT	=	"HTTP"

**Variables**

Name	Type	Value
errorCode	Extract	MSGTEXT.Response Code: <*.code>\$.
lookupPrefix	Constant	"HTTP_"
lookupCode	Combine	lookupPrefix,errorCode.code
message	Lookup	lookupCode
desiredRange	Constant	"5<25>"

**Advanced Filter**

Name	Operator	Value
errorCode.code	matches	desiredRange

**Parameters**

Want Original ☐ Enhance Always ☒

Buttons: OK, Cancel, Help

- 12 Alter the message:
  - a Click the **New Alarms** tab.  
The New Alarm panel opens.
  - b From the drop-down menu, select **Alter Specification**.  
The Alter Alarm Definition table displays.
  - c Define the following attributes:
    - From the Field drop-down menu, select **MSGTXT**.
    - From the Mode drop-down menu, select **Append**.
    - From the Value drop-down menu, select **message**.
- 13 Click **OK** to complete the definition of the correlator.

## Enriching Messages by Using Perl Commands

As an example of Enhance correlation, you might want to use Perl commands to enhance messages, as follows:

- [Appending Comment Fields to Message Text](#) on page 206
- [Adding a CMA by Name to an Event](#) on page 207
- [Increasing Event Severity for Non-Critical Events](#) on page 208
- [Determining Whether To Suppress Events Based on Maintenance Mode](#) on page 209

## Appending Comment Fields to Message Text

You can use a Perl script to return data to Composer that can then be added to a message. In this example, a message is enhanced by appending the comment fields for a server `/etc/hosts` entry to the message text.



On Windows, the host file is `%SystemRoot%\system32\drivers\etc\hosts`. This Windows file has the same format as in UNIX.

This implementation assumes that `/etc/hosts` has the following format:

```
IP_address hostname # Comments
```

For example, the file could contain the following function:

```
1.2.3.4 server_4.company.net # Asset ID 12345
```

This function would return the following:

```
" Asset ID 12345"
```

To append any message text with the comment fields from `/etc/hosts`, you could use the Perl command `getHostInfo`, as follows:

```
sub getHostInfo {  
  
    local($nodename)=@_  
    local($infile);  
    local($line);  
  
    if ( $^O eq "MSWin32" )  
    { $infile = $ENV{ SystemRoot } .'\System32\drivers\etc\hosts';  
    }  
    else # Unix platforms  
    { $infile = "/etc/hosts"; }  
  
    if (!open (INFILE, "<$infile") )  
    { print "Error while opening the hosts file"; exit; }  
  
    while (<INFILE>)  
    {  
        $line = $_  
        if( $line =~ /$nodename.*\#(.*)/ )  
        { return "  $1"; }  
    }  
    return "  ";  
}
```



This portable Perl code should work on all supported platforms.

The correlation requirement is to call `getHostInfo(msg_node_name)` and add the return value to the message text.

## Adding a CMA by Name to an Event

You can use a Perl script to read a custom message attribute (CMA) from an external file and add it to an event. In this example, you identify a team to work on the event from a file containing this mapping, based on an event's message group. In this example, the mapping file is named `teams.txt`.

By default, the `teams.txt` file is located in the following directory:

- *UNIX*  
`$OV_CONTRIB/ecs/external/perl`
- *Windows managed node*  
`%OvInstallDir%\contrib\ecs\external\perl`
- *Windows management server*  
`%OvDataDir%\contrib\ecs\external\perl`

The `teams.txt` file contains entries in the following format:

Message\_group TeamName

The file might contain entries such as the following:

```
SNMP    NetAdmins
OpC     OpsTeam
VP_SM   OpsTeam
...
```

This Perl subroutine returns the `TeamName` string to the correlator that invokes the `getTeamInfo()` function:

```
sub getTeamInfo {
    local($msggrp)=@_;
    local($infile) = "teams.txt";
    local($line);
    if (!open (INFILE, "<$infile")) { ... }
    while (<INFILE>)
    {
        $line = $_;
        if( $line =~ /$msggrp\t(.*)/ )
        { return " $1"; }
    }
    return " ";
}
```

To set a CMA named `CMA_TEAM_NAME` to reflect the `getTeamInfo()` return string, in an Enhance correlator, you would define a variable `teaminfo` of the type `Function` with a value of `getTeamInfo(GROUP)`. Then, in the correlator's Alarm Definition window, you could alter the alarm's definition by adding a field `CMA_TEAM_NAME`, in mode `Replace`, with a value of `teaminfo`. This command requires `CMA_TEAM_NAME` to be added to the `CO.conf` file.

## Increasing Event Severity for Non-Critical Events

You can use external Perl functions to modify an event's severity. The code sample in this section shows how to increase the severity by one, where possible. Notice that event severity is represented internally as an integer value. Notice also how the severities and their integer values are not in a numerical order, thus requiring some special handling when manipulating them programmatically:

```
sub incrSeverity {
    local($origSev) = @_;
    local($newSev);
    local($SEV_NORMAL)    = 8;
    local($SEV_WARNING)   = 16;
    local($SEV_MINOR)     = 64;
    local($SEV_MAJOR)     = 128;
    local($SEV_CRITICAL)  = 32;

    if ($origSev >= $SEV_NORMAL) {

        $newSev = $origSev * 2;

        if ($newSev == 256)
        { $newSev = $SEV_CRITICAL;
          return int $newSev; }
        if ($newSev == $SEV_MINOR)
        { $newSev = $SEV_CRITICAL;
          return int $newSev; }
        if ($newSev == $SEV_CRITICAL)
        { $newSev = $SEV_MINOR;
          return int $newSev; }
    }
    else { $newSev = $origSev; }
    return int $newSev;
}
```



To apply a function to change an event's severity, you would follow these steps:

- 1 Define a new variable (for example, by the name `NewSeverity`) of the type Function and with a value of `incrSeverity(SEVERITY)`.
- 2 In the correlator's Alter Alarm Definition section, add an entry for the field `SEVERITY`, in Replace mode, with a value of `NewSeverity`.

## Determining Whether To Suppress Events Based on Maintenance Mode

Many users need to flexibly manage planned or unplanned outages for the managed node. One goal of the outage management could be to suppress all events from nodes during an outage. A very simple implementation approach would be to maintain a file with the hostnames for nodes that are considered to be in an outage. (This file would be maintained by an external mechanism.) You could query the external file for the name of the message host, and return different values, based on whether the hostname was found in the outage file. A Suppress correlator could then use the function's return to determine whether an event needed to be suppressed.

To determine whether to suppress events based on maintenance mode, you could use the Perl command `check_outage`, as follows:

```
sub check_outage {  
  
    my ($node) = @_ ;  
    open MAINT, "/some/path/to/maint/file";  
    while (<MAINT>) {  
        chomp;  
        if (/^$node$/o) {  
            close MAINT;  
            return(1);  
        }  
    }  
    close MAINT;  
    return(0);  
}
```

If a host is listed in the external file, the event is suppressed, based on the return of a Perl function. The function is used in a Advanced Filter definition for the Suppress correlator.

## Case 2: Suppress Correlation

In the HPOM environment, you use Suppress correlation to filter out messages. You can also use the standard HPOM Message Source template conditions. However, it is sometimes easier to set simple message conditions in the template definitions, and use the correlator to suppress an unwanted subset of messages.

### Suppressing Message Subsets

As an example of enhance correlation, you might want to suppress an unwanted subset of messages. This example is an extension of the HTTP server message enrichment described in [Replacing Error and Status Codes with Descriptions](#) on page 200. In the Enhance correlator, you can replace the error or status code with a description if the codes are in the range of 500 through 599. For codes outside this range, you can use Suppress correlation.

This kind of Suppress correlation verifies that the OBJECT message attribute is set to HTTP, and then finds the error or status code in the MSGTEXT. If the error code is not in the 500 through 599 range, it is suppressed.

Consider the following messages:

```
NODENAME:      system1
OBJECT:        HTTP
APPLICATION:    Web Monitor
MSGTEXT:       Response Code: 502
```

```
NODENAME:      system1
OBJECT:        HTTP
APPLICATION:    Web Monitor
MSGTEXT:       Response Code: 401
```

The first message should be passed and enriched by Enhance correlation. The second message is suppressed.

# Suppressing the Subset of a Message

To suppress the subset of a message, you would follow these steps:

1 **Identify the appropriate messages.**

All HTTP messages are identified by the following attributes:

- OBJECT is set to "HTTP".
- MSGTEXT has format such as "Response Code: 502".

2 **Decide whether messages participate in other correlations.**

Table 13 summarizes the options available in the Suppress Correlator Template window.

**Table 13 Buttons in the Suppress Correlator Template Window**

Button Name	Selected	Functionality	When to Use
Participate In Other Correlation	No	Alarm does not participate in other correlations before it is discarded.	Default option. Suppress messages before they could be seen by other correlations.
	Yes	Alarm takes part in other correlations before it is discarded.	Typically, you use this option if the message contributes to another correlation logic, but itself is not output.

For this example, if you decided to suppress a message (not in the 500 through 599 code range), you would not enhance it. As a result, you could safely suppress it and not have it participate in other correlations.

## Defining the Suppress Correlator Template

To define the Suppress correlator template, follow these steps:

- 1 Do one of the following:
  - From the Correlator Store window, click **Correlations**→**Correlator Templates**→**Suppress**.



- Click the **Suppress** button.



The Suppress Correlator Template window opens.

- 2 In the Name text box, type a name for the correlator.
  - ⚠ You cannot use spaces in the correlator name.
- 3 In the Description text box, type a description of the correlator.



- 4 Click the **Definition** tab.

- 5 Define the Alarm Signature:
  - a In the Field cell, select **OBJECT**.
  - b From the Operator field drop-down menu, select the equal sign (=).
  - c In the Value field, enter **"HTTP"**.
- 6 Create an `errorCode` variable to hold the error or status code extracted with a pattern from the message text:
  - a In the Name cell, type **errorCode**.
  - b From the Type field drop-down menu, select **Extract**.
  - c In the Attribute drop-down menu, select **MSGTEXT**.
  - d In the Pattern drop-down menu, select **Response Code: <\*.code>\$**.



You do not need to specify the pattern separator if you use normal whitespace as the word separator pattern. Later, you can use the `code` tag to represent that part of the extracted the message text (the HTTP status or error code).

► In Composer, you use the pattern matching syntax used in HPOM for the UNIX template configuration.

- 7 Create a `desiredRange` variable to hold the pattern used in the Advanced Filter to select only messages that are not for HTTP code or status values in the range 500 through 599:
  - a In the Name cell, type **desiredRange**.
  - b From the Type field drop-down menu, select **Constant**.
  - c In the Value field, enter **"5<2#>"**.
- 8 Declare the Advanced Filter entry, `errorCode.code`, to match the pattern declared in `desiredRange`:
  - a In the Name cell, select **errorCode→code**.
  - b From the Operator field drop-down menu, select **does NOT match**.
  - c In the Value field, select **desiredRange**.

You use the Advanced Filter to select a subset of messages that match the Alarm Signature. Only those message are suppressed.

At this point, the Definition tab should look like this.

**Suppress**

Name:

**Definition** (selected) | Description | Cancel

**Alarm Signature**

Field	Operator	Value
OBJECT	=	*HTTP*

**Variables**

Name	Type	Value
errorCode	Extract	MSG-TEXT, "Response Code: <#.code>#"
desiredRange	Constant	"5<2#>#"

**Advanced Filter**

Name	Operator	Value
errorCode.code	does NOT match	desiredRange

**Parameters**

Participate in Other Correlation ☒

OK Cancel Help

- 9 Click **OK** to complete the definition of the correlator.

## Case 3: Multi-Source Correlation

In the HPOM environment, you use Multi-Source correlation to detect and suppress messages caused by another fault or situation. These messages are also known as sympathetic messages. Sometimes, you may want to simply suppress the sympathetic messages. At other times, you may also want to generate a new message that shows a greater problem or situation.

Examples of Multi-Source correlation include the following:

- Core router goes down on a remote site, and HPOM generates alerts indicating that a number of agents are not responding.
- File system full messages—as well as OS SPI, application, and database writing errors—are generated.
- DNS issues, as well as the system running DNS, are detected.

### Suppressing Messages on Remote Sites

An example of Multi-Source correlation is a situation in which the primary router for a remote site goes down. There are a number of systems within that site that may be detected as down or unreachable from heartbeat monitoring.

In this situation, if you received a router down message, you would suppress any HP Operations Agent Not Responding messages that came from that site (based on device or node naming conventions). You would also report the site as down.

Suppressing messages on remote sites involves the relative timing of the messages.

Consider the following five messages:

- 1 Message 1 is issued:

```
NODENAME:      crt01siteA
MSGTYPE:       OV_APA_NODE_DOWN
MSGTEXT:       Node Down
```

This message should be output immediately because it is a Node Down message from a router (using the `ctr` naming prefix for core router).

2 Message 2 is issued 1 minute after Message 1:

```
NODENAME:      sys05siteB
MSGTEXT:       OpC agent not responding.
```

This message should be output because it comes from a different site (siteB) than Message 1 (siteA).

3 Message 3 is issued 10 seconds after Message 2:

```
NODENAME:      sys05siteA
MSGTEXT:       OpC agent not responding.
```

This message should be suppressed because it is from the same site as Message 1. The message should trigger a Site Down message because it is an Agent Not Responding message.

4 Message 4 is issued 1 second after Message 3:

```
NODENAME:      sys03siteA
MSGTEXT:       OpC agent not responding.
```

This message should be suppressed because it is from the same site as Message 1. However, it should *not* trigger another Site Down message.

5 Message 5 is issued 20 minutes after Message 4:

```
NODENAME:      sys04siteA
MSGTEXT:       OpC agent not responding.
```

This message should be output because it occurs too long after the original scenario was triggered, even though it is an Agent Not Responding message from the same site.

## Suppressing a Sympathetic Message on a Remote Site

To suppress a sympathetic message on a remote site, you would follow these steps:

1 **Identify the appropriate messages.**

All Core Router Down messages are identified by the following:

- MSGTYPE is OV\_APA\_NODE\_DOWN.
- NODENAME starts with "crt" (for core router).
- Location is extracted from NODENAME (after the digits).



All Agent Not Responding messages are identified by the following:

- MSGTEXT is OpC agent not responding.
- Locations are extracted from the NODENAME (after the digits).

You must allow up to 10 minutes for all messages to be received.

## 2 Determine which other parameters to consider.

[Table 14](#) summarizes the options available for Multi-Source correlations.

**Table 14 Buttons in the Multi-Source Correlator Template Window**

Button Name	Selected	Functionality	When to Use
Discard on Set Completion	No	Alarm is forwarded, regardless of set completion.	Default option. Typically, you select this option for messages that are the root cause or trigger of the scenario.
	Yes	Alarm is discarded if the set is complete. Otherwise, it is forwarded.	Typically, you select this option for the sympathetic messages in the scenario.
Window Period	N/A	Mandatory field. Time period within which all alarms of the set must arrive for the set to be considered complete. The alarms can arrive in any order.	Always required. Set a realistic time period within which all messages must arrive for the given scenario. <b>NOTE:</b> All messages that are matched are delayed for this window period, regardless of the result of the correlation.
Set	No	Operates in Mode 1. For details, see <a href="#">Multi-Source Correlator Template</a> on page 32.	Default mode. Typically, you use this option for all scenarios in which there can be a number of sympathetic messages of the same type to be suppressed.
	Yes	Operates in Mode 2. For details, see <a href="#">Multi-Source Correlator Template</a> on page 32.	Typically, you use this option when there is a fixed set of messages that determine an issue. After the issue has been established, no extra messages are suppressed.

## Suppressing Subsets with the Multi-Source Correlator Template

To suppress subsets with the Multi-Source correlator template, follow these steps:

- 1 Do one of the following:
  - From the Correlator Store window, click **Correlations**→**Correlator Templates**→**Multi-Source**.



- Click on **Multi-Source** button.



The Multi-Source Correlator Template window opens.

- 2 In the Name text box, type a name for the correlator.



You cannot use spaces in the correlator name.

- 3 In the Description text box, type a description of the correlator.



Unlike previous examples in this chapter, multiple message (alarms) need to be configured for Multi-Source correlations. You need to complete the details of each message (alarm) before moving on to the next.

- 4 Click the **Definition** tab.
- 5 Enter a name for the first message (alarm) type.

In the Name text box, type **routerDown**.



- 6 Define the Alarm Signature:
  - a In the Field cell, select **MSGTYPE**.
  - b From the Operator field drop-down menu, select the equal sign (=).
  - c In the Value field, enter "**OV\_APA\_NODE\_DOWN**".
- 7 Create a device variable to hold the location code to be extracted with a pattern from the node name (device):
  - a In the Name cell, type **device**.
  - b From the Type drop-down menu, select **Extract**.
  - c In the Attribute drop-down menu, select **NODENAME**.
  - d In the Pattern text field, type "**^<\*><#><\*.location>**".



- 8 Create a coreRouter variable for the Advanced Filter that selects only messages from nodes that match the naming convention for core routers:
  - a In the Name cell, type **coreRouter**.
  - b From the Type field drop-down menu, select **Constant**.
  - c In the Value field, type "**^<cr>**".

You do not need to specify the pattern separator if you use normal whitespace as the word separator. Later, you can use the location tag to represent that part of the extracted node name (the site name).

- 9 Create a message variable for the message text for the Site Down message that is generated on detection:

- a In the Name cell, type **message**.
- b From the Type field drop-down menu, select **Constant**.
- c In the Value field, type **"Site Down"**.

The Advanced Filter is used to select a subset of messages that match the Alarm Signature (Node Down), and only those of interest (nodes that are core routers).

- 10 Declare the Advanced Filter entry to match the pattern declared in the `coreRouter` variable:

- a In the Name cell, select **routerDown->NODENAME**.
- b From the Operator field drop-down menu, select **matches**.
- c In the Value field, select **routerDow->coreRouter**.

- 11 In the Message Key cell, select **routerDown->device->location**.

You use the message key to match the different message (alarm) types. In this case, you match the site (location) name extracted from `NODENAME`.



Do not select the Discard on Set Completion – Router Down check box. You do not want to suppress root cause messages.

- 12 Under Window Period, type the following:

- **00** hours (hh)
- **10** minutes (mm)
- **00** seconds (ss)

The Window Period is a global setting used for this correlator. It defines the time period in which messages are related.

The definition of the routerDown component now looks like this.

The screenshot shows the 'routerDown' alarm signature configuration window. The 'Name' field is 'SiteOutage'. The 'Definition' tab is active, showing the following sections:

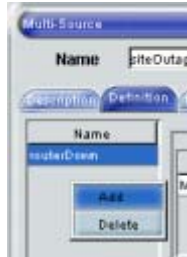
- routerDown - Alarm Signature:** A table with one row: MSOTYPE = "OV APA NODE DOWN".
- routerDown - Variables:** A table with three rows:
 

Name	Type	Value
device	Extract	NODENAME, ^c"><@>c".location>\$,
coreRouter	Constant	"^c"
message	Constant	"Site Down"
- routerDown - Advanced Filter:** A table with one row:
 

Name	Operator	Value
routerDown.NODENAME	Matches	routerDown.coreRouter
- routerDown - Message Key:** A text field containing 'routerDown.device.location'.
- routerDown - Alter Alarm Definition:** An empty table with columns: Field, Mode, Value.

At the bottom, there is a checkbox for 'Discard on Set Completion' and a 'Parameters' section with a 'Window Period' set to '00 hh 10 mm 00 ss' and a 'Set' button. At the very bottom are 'OK', 'Cancel', and 'Help' buttons.

- 13 Add another message alarm for the Agent Not Responding message:
  - a In the Name section, right-click the **routerDown** entry.



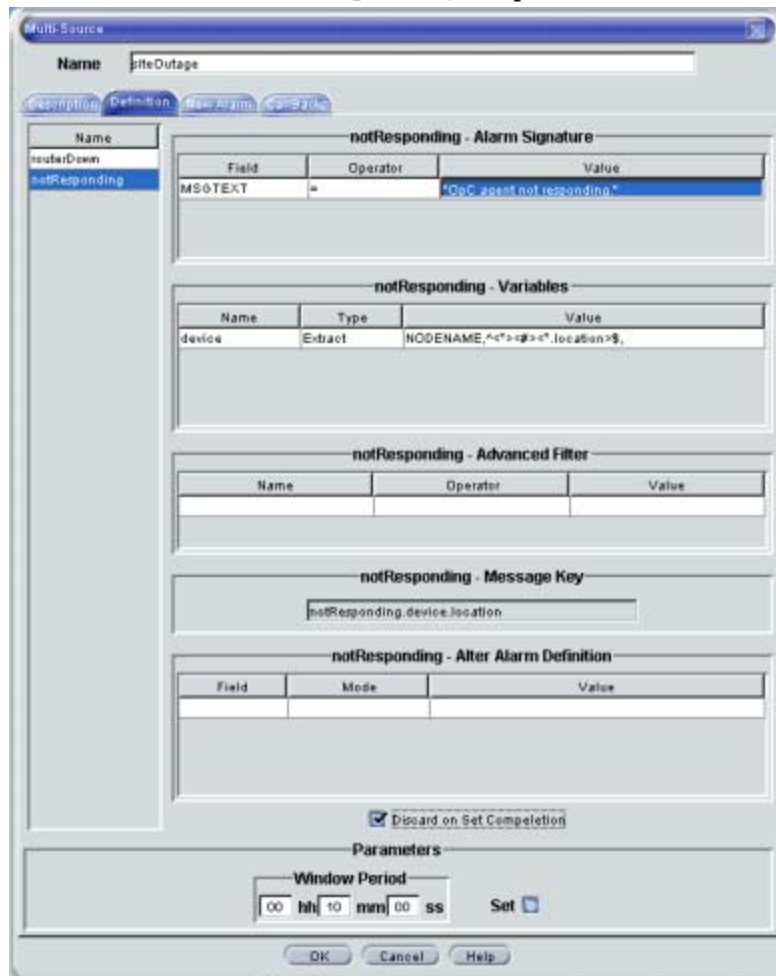
- b In the Name field, type **notResponding**.



A blank entry appears for the definition of the Agent Not Responding message.

- 14 Enter the following values to define the Alarm Signature (to select only the Node Down messages):
  - a In the Field cell, select **MSGTEXT**.
  - b From the Operator field drop-down menu, select the equal sign (=).
  - c In the Value field, type the following:  
**"OpC agent not responding"**
- 15 Create a device variable to hold the location code to be extracted with a pattern from the node name (device):
  - a In the Name cell, type **device**.
  - b From the Type field drop-down menu, select **Extract**.
  - c In the shortcut menu from the Value field, select the following:
    - Attribute: **NODENAME**
    - Pattern: **^<\*><#><\*.location>**

The definition of the `notResponding` component now looks like this.



- 18 Click the **New Alarms** tab to create the Site Down message.  
The New Alarm panel opens.
- 19 From the drop-down menu, select **Alter Specification**.  
The Alter Alarm Definition table displays.
- 20 Define the following attributes to alter the new alarm:
  - a From the Field drop-down menu, select **NODENAME**.
  - b From the Mode drop-down menu, select **Replace**.
  - c From the Value drop-down menu, select **routerDown->NODENAME**.
- 21 Add another fields line:
  - From the Field drop-down menu, select **MSGTEXT**.
  - From the Mode drop-down menu, select **Replace**.
  - From the Value drop-down menu, select **routerDown->message**.

The New Alarm tab now looks like this.

Field	Mode	Value
NODENAME	Replace	routerDown.service.location
MSGTEXT	Replace	routerDown.message

- 22 Click **OK** to complete the definition of the correlator.



## Case 4: Rate Correlation

In the HPOM environment, you use Rate correlation to handle scenarios where it is necessary to detect issues, based on the frequency of messages.

You might want to detect issues, based on message frequency, in the following situations:

- Issue does not exist unless a number of the same messages occur within a short time period.
- Severity is increased when a certain rate of the same message occurs.

### Detecting DNS Outages

Sometimes, a Domain Name Server (DNS) does not respond. If this lack of response occurs only occasionally, it may be part of normal operations. However, if a DNS does not respond repeatedly within a short period of time, the problem may be serious enough to require attention. One example of a serious problem is a DNS outage.

In response to a DNS outage, you verify that the OBJECT message attribute is set to DNS, and that the MSGTEXT contains the following:

```
"No response from server Name Server"
```

If you receive five such messages from the same NODENAME within 10 seconds, you generate a new message indicating a likely DNS outage.

Consider the following messages, which occur within 10 seconds:

```
NODENAME:      system1
OBJECT:         DNS
MSGTEXT:        No response from server Name Server
```

```
NODENAME:      system2
OBJECT:         DNS
MSGTEXT:        No response from server Name Server
```

```
NODENAME:      system1
OBJECT:         DNS
MSGTEXT:        No response from server Name Server
```

```
NODENAME:      system1
OBJECT:        DNS
MSGTEXT:       No response from server Name Server
```

```
NODENAME:      system1
OBJECT:        DNS
MSGTEXT:       No response from server Name Server
```

```
NODENAME:      system1
OBJECT:        DNS
MSGTEXT:       No response from server Name Server
```

For system1 and system2, you should suppress all messages. On the arrival of the fifth occurrence of the message from system1, you generate a new message. For system2, you generate nothing.

## Responding to a DNS Outage

To respond to a DNS outage, you would do the following:

- 1 **Identify the appropriate messages.**

All DNS server problem messages are identified by the following:

- OBJECT is set to "DNS".
- MSGTEXT has a format such as the following:

```
"No response from server Name Server"
```

- 2 **Decide whether to discard contributing messages.**

There is no issue until the frequency of the messages from the same node breaches a threshold. For this reason, you should discard the contributing messages. In this example, you suppress all of the contributing messages if the rate threshold is breached. That is, you select the **Discard** flag.

Table 15 lists the options available in the Rate Correlation Template window.

**Table 15 Buttons in the Rate Correlator Template Window**

Button Name	Selected	Functionality	When to Use
Window Period	N/A	Time period for which the message arrival rate is monitored	Mandatory field. Must always be set.
Count	N/A	Threshold count. If the number of messages exceeds the threshold count within the specified window period, the rate threshold is breached.	Mandatory field. Must always be set.
Discard	No	Messages are not discarded. If created, the new message is output.	Default mode. In HPOM environments, this mode is <i>not</i> used very often. Use this option when you are required to output all of the messages that led to the triggering of the issue. Normally, you use this option when the contributing messages provide information that augments the triggered message.
	Yes	All messages are discarded. Only the new message, if created, is output.	In HPOM environments, this is the most commonly used mode. All contributing messages are discarded because there is no problem until the frequency threshold is breached (and a new message is generated).

## Defining the Rate Correlator Template

To define the Rate correlator template, follow these steps:

- 1 Do one of the following:
  - From the Correlator Store window, select **Correlations**→**Correlator Templates**→**Rate**.



- Click the **Rate** button.



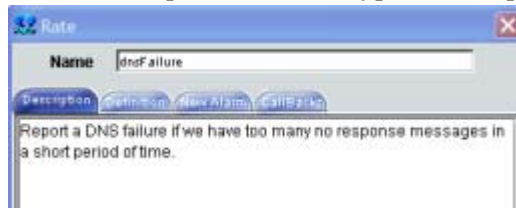
The Rate Correlator Template window opens.

- 2 In the Name text box, type a name for the correlator.



You cannot use blank spaces in the correlator name.

- 3 In the Description text box, type a description of the correlator.



- 4 Define the Alarm Signature to select only the messages of interest:
  - a Click the **Definition** tab.
  - a In the Field cell, select **OBJECT**.
  - b From the Operator field drop-down menu, select the equal sign (=).
  - c In the Value field, enter **"DNS"**.

For all messages in which the OBJECT is DNS, you use the Advanced Filter to select only those messages that are considered DNS No Response messages.

- 5 Create an `errorString` variable to hold the generated message text:
  - a In the Name cell, type **`errorString`**.
  - b From the Type field drop-down menu, select **Constant**.
  - c In the Value field, type the following:  
**"Likely DNS Failure"**
- 6 Create a `msgText` variable to hold the pattern to be used in the Advanced Filter to select only messages of interest:
  - a In the Name cell, type **`msgText`**.
  - b From the Type field drop-down menu, select **Constant**.
  - c In the Value field, type the following:  
**"No response from server Name Server"**
- 7 Declare the Advanced Filter entry:
  - a In the Name cell, select **MSGTEXT**.
  - b From the Operator field drop-down menu, select **matches**.
  - c In the Value field, select **`msgText`**.

You use the Advanced Filter to select a subset of messages that match the Alarm Signature. Only these messages are used to test the frequency of the occurring messages.

- 8 In the Message Key cell, select **NODENAME**.

You use the message key to match the messages from the same source. In this case, you use the message key to count the messages for each node on which they occur.

- 9 Under Window Period, set the following:

- **00** hours (hh)
- **00** minutes (mm)
- **10** seconds (ss)

The window period is a global setting used for this correlator. It defines the time period in which related messages are matched.

- 10 In the Count text box, type **5**.

The count is the number of messages from the same source (message key) that must be received within the specified window period for a rate or frequency threshold breach to occur.

- 11 Select the **Discard** check box to discard the contributing messages.

The Discard flag determines whether messages contributing to the rate check are suppressed (discarded).

The Rate configuration now looks like this.

The screenshot shows the 'Rate' configuration window with the following sections:

- Name:** dns failure
- Alarm Signature:**

Field	Operator	Value
OBJECT	=	"DNS"
- Variables:**

Name	Type	Value
emerString	Constant	"Likely DNS Failure"
msgText	Constant	"No response from server Name Server"
- Advanced Filter:**

Name	Operator	Value
MSGTEXT	matches	msgText
- Message Key:** NODENAME
- Parameters:**
  - Window Period:** 00 hh 00 mm 10 ss
  - Count:** 5
  - Discard:** ☒

Buttons at the bottom: OK, Cancel, Help.

- 12 Alter the alarm:
  - a Click the **New Alarms** tab.  
The New Alarm panel opens.
  - b From the drop-down menu, select **Alter Specification**.

The Alter Alarm Definition table displays.

Rate

Name: dnsFailure

Description Definition **New Alarm** CallBacks

Alter Specification ▼

**Alter Alarm Definition**

Field	Mode	Value
MSGTEXT	Replace	errorString

☐ Feedback

OK Cancel Help

- c Define the following attributes:
  - From the Field drop-down menu, select **MSGTXT**.
  - From the Mode drop-down menu, select **Replace**.
  - From the Value drop-down menu, select **errorString**.
- 13 Click **OK** to complete the definition of the correlator.



## Case 5: Transient Correlation

Transient correlation matches messages in transient pairs.

A transient pair is made up of the following messages:

- Set  
Typically indicates an error condition.
- Clear  
Signifies the end of that same error condition.

This pair of related messages is considered transient when it occurs within a predefined time interval (typically relatively short).

Often, when Set or Clear message pairs occur within a short period of time, they represent only a fleeting problem and can safely be ignored. If too many transient matches occur from the same source in a given time period, you can use Transient correlation to generate a new message.

### Generating New Messages

You might want to generate a new message in the following situations:

Some common situations where this would occur include the following:

- Messages
  - Application Down
  - Application Up
  - Interface Down
  - Interface Up
  - Node Down
  - Node Up
  - Service Down
  - Service Up
- CPU utilization problems
- Network link flapping (that is, multiple transients in a short time period)

## Smart Message Correlation

HPOM includes templates, based on smart message correlation (that is, the state-based browser). These templates provide functionality similar to that of the Transient correlator.

Smart message correlation and Transient correlation have the following key differences:

- **Set messages**

Transient correlation completely suppresses a `Set` message if it receives a matching `Clear` message. In the same situation, smart message correlation outputs the `Set` message and later acknowledges it.

- **Flapping conditions**

You can use Transient correlation to detect a possible flapping condition (repeated `Set` and `Clear` pairs from the same source).

- **Message matching**

With the state-based browser, you can match messages as `Set` and `Clear` pairs that have been apart for a long time, even after a software restart. In contrast, Composer relies on short-term, memory-based window matching.

Despite these key differences, smart message correlation and Transient correlation can have similar outcomes. The key differences can help you to determine which mechanism is most appropriate for your needs.

## Suppressing High CPU Utilization Messages

A high CPU utilization message may indicate a “rogue” process running on a system. More commonly, the message is followed shortly by a message indicating that the issue no longer exists. In such a situation, you can safely suppress both messages. You should output the high CPU utilization message only if the `Clear` message does not occur within a short period of time.

Consider the following messages, which occur within 10 seconds:

```
NODENAME:      system1
MSGTEXT:       system1:LZ has high CPU usage.
```

```
NODENAME:      system1
MSGTEXT:       system1:LZ no longer has high CPU usage.
```

You should suppress both messages because the problem no longer exists.

## Responding to High CPU Utilization Messages

To respond to high CPU utilization messages, you would follow these steps:

### 1 Identify the appropriate messages.

CPU utilization messages are identified by the following:

- MSGTEXT for the Set message has a format such as the following:  
"system1:LZ has high CPU usage."
- MSGTEXT for the Clear message has a format such as the following:  
"system1:LZ no longer has high CPU usage."

### 2 Decide how to match the messages.

Transient pairs of Set and Clear messages are identified by the following:

- Have the same NODENAME value.
- Occur within 10 seconds of each other. (Set is before Clear.)

**Table 16** summarizes the options available in the Transient Correlation Template window.

**Table 16 Buttons in the Transient Correlator Template Window**

Button Name	Selected	Functionality	When to Use
Window Period	N/A	Mandatory field. The maximum time a Set message is held by Composer while waiting for a Clear message. If the Clear message is received while the alarm is held, both the Set and Clear messages are discarded. If no Clear message is received in this window period, the Set message is output.	Mandatory field. Must always be set. Typically, the period is somewhere between 10 seconds and 10 minutes, depending on the severity of the problem.
Enable Threshold	No	No Count is maintained. The Threshold Count and Threshold Windows are both disabled.	Default option. This option is applicable to situations in which a flapping (multiple transients in a short time period) condition is not likely or is not appropriate to detect and report.
	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.	This option is applicable in situations where a flapping (multiple transients in a short time period) condition is possible, and where it is appropriate to detect and report the condition.
Threshold Count	N/A	Threshold count. If the number of transient pair matches for the same source exceeds the threshold count within the specified window period, the rate threshold is breached.	This field is enabled only if the Enable Threshold button is enabled.
Threshold Window	N/A	Time period for which the count is maintained.	This field is enabled only if the Enable Threshold button is enabled.

## Defining the Transient Correlator Template

To define the Transient correlator template, follow these steps:

- 1 Do one of the following:
  - From the Correlator Store window, select **Correlations**→**Correlator Templates**→**Transient**.



- Click the **Transient** button.



The Transient Correlator Template window opens.

- 2 In the Name text box, type a name for the correlator.



You cannot use blank spaces in the correlator name.

- 3 In the Description text box, type a description of the correlator.



- 4 Define the Alarm Signature (to select only the messages of interest):
  - a Click the **Definition** tab.
  - a In the Field cell, select **MSGTEXT**.
  - b From the Operator field drop-down menu, select **matches**.
  - c In the Value field, type **"high CPU usage"**.

In this step, you consider all messages in which the MSGTEXT contains high CPU usage.



The Alarm Signature must match for both Set and Clear messages.

- 5 Create a clearMessage variable to hold the text to match for Clear messages:
  - a In the Name cell, type **clearMessage**.
  - b From the Type field drop-down menu, select **Constant**.
  - c In the Value field, type **"no longer"**.



In this scenario, nothing is required for the Advanced Filter.

- 6 In the Message Key cell, select **NODENAME**.

You use the message key to match messages from the same source.

- 7 Under Window Period, type the following:
  - **00** hours (hh)
  - **00** minutes (mm)
  - **10** seconds (ss)

The window period is a global setting used for this correlator. It defines a time period in which transient messages are matched.



Do *not* select the Enable Threshold check box because you are not checking for a flapping condition. The Threshold Count and Threshold Windows fields are inactive.

- 8 Set the matching criteria for the Clear messages:
  - a Click the **Clear Alarm** button.
  - b In the Attribute cell, select **MSGTEXT**.
  - c From the Operator field drop-down menu, select **matches**.
  - d In the Value cell, type **clearMessage**.

At this point, the Transient configuration looks like this.

- 9 Click **OK** to complete the definition of the correlator.





## 9 Developer Mode in NNM

This chapter explains how developers manage HP Correlation Composer in HP Network Node Manager (NNM) environments:

- [Administrative Tasks](#) on page 241
- [Starting Composer in Developer Mode](#) on page 242
- [Configuring Operator Profiles](#) on page 242
- [Defining Operator Access](#) on page 255
- [Deploying the Correlator Store](#) on page 257



Unless otherwise noted, all tasks described in this chapter should be executed by the Correlator Store developer.

### Administrative Tasks

In Composer, developers are responsible the following tasks:

- **Correlator Stores**

Creating and modifying the Correlator Store file. For instructions, see [Chapter 4, Developing Correlators](#).

- **Operator access**

Defining operator access by maintaining Namespace and Security files.

- **Correlation logic**

Creating the Deploy Configuration file that deploys the correlation logic to the HP Event Correlation Services (ECS) engine.

# Starting Composer in Developer Mode

To start Composer in Developer mode, type the following:

```
ovcomposer -m d
```

For more information about the `ovcomposer` command, see the *ovcomposer* reference page.

## Configuring Operator Profiles

To configure operator profiles for Composer, complete the following tasks:

- Task 1: [Creating Correlator Stores](#) on page 242
- Task 2: [Listing Correlator Stores](#) on page 242
- Task 3: [Creating NameSpace and Security Files](#) on page 242
- Task 4: [Creating Deploy Configuration Files](#) on page 251

After you plan operator profiles, you can provide them with access rights. For details, see [Defining Operator Access](#) on page 255.

### Creating Correlator Stores

As a developer, you create Correlator Store files in a way that logically groups correlators defined for set environments. That is, you put all correlators defined for one environment into one Correlator Store. For details, see [Chapter 4, Developing Correlators](#).

### Listing Correlator Stores

As a developer, you must give operators access to the Correlator Store files that display correlation logic. To do so, you create a list of Correlator Stores to be displayed to operators.

### Creating NameSpace and Security Files

As a developer, you create NameSpace and Security files for operators.

## Namespace Files

A Namespace file contains a list of Correlator Store files, grouped logically to define operator profiles. You use this list to assign access permissions for the Correlator Store files to operator profiles. The list has no other purpose. The list of Correlator Stores specified in this file determines the area of operation within which an operator can work.

A Namespace file is a simple ASCII file you can edit in any standard text editor. This file lists named value pairs of the Correlator Store name and the relative path of the Correlator Store location.



To create or edit a Namespace file, you need `root` access on the machine where Composer is installed.

### Syntax of the Namespace File

Namespace files use the following general syntax:

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

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

This syntax includes the following parameters:

```
<Logical Name1>
```

```
<Logical Name2>
```

Logical names of Correlator Store files. These names are displayed in Composer when started in Operator mode.

```
<Location of the Correlator Store file>
```

Location of the Correlator Store file relative to the following directory:

- *UNIX*

```
$OV_CONF/ecs/CIB
```

- *Windows*

```
%OV_CONF%\ecs\CIB
```

### Example of a Namespace File

A Namespace might look like this:

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

## Guidelines for NameSpace Files

When creating NameSpace files, follow these guidelines:

- **Locations**

Make sure the location of the Correlator Store file on the right side of the equal sign (=) is always relative to the following directory:

- *UNIX*

`$OV_CONF/ecs/CIB`

- *Windows*

`%OV_CONF%\ecs\CIB`

You may *not* add Correlator Store files above this directory. The files must be in this directory or in a subdirectory (typically, with the same name as that of the Correlator Store).

- **Spaces**

Do not add a blank space before or after the equal sign (=).

- **Names**

Use unique logical names for Correlator Stores. Place every entry for a logical name on a separate line.

- **Paths**

Specify all file location paths on a single line.

- **Extensions**

Always save NameSpace files with the `.conf` extension.

- **Comments**

Precede all comments with the number sign (#).



Make sure the NameSpace file referenced in the Deploy Configuration file (see [Creating Deploy Configuration Files](#) on page 251) is the same file passed with the `-N` option when Composer is started (see the *ovcomposer* reference page). If the file names are different, you create one set of Correlator Stores and then deploy a completely different set of Correlator Stores.

## Security File

In Composer, the Security file contains a list of fields and parameters that can be edited by operators. Each Correlator Store file has a corresponding Security file associated with it. The Security file is stored in the same directory as the Correlator Store file.

*Chapter 9*

The first time you save a Correlator Store file, a default Security file is created:

```
<Correlator Store filename>.sec
```

In this file name, *<Correlator Store filename>* is the name of the Correlator Store for which the Security file is created.

The default Security file enables operators to edit all parameter values in the Alarm Definition section of all correlators. The Security file also contains a list of correlator fields that operators can edit. Only the values of these fields can be edited by the operators.

The Security file is a simple ASCII file that can be edited in any standard text editor.

### Syntax of the Security File

Security files use the following general syntax:

```
ALL_TEMPLATE=TOK_LIST
```

```
ALL_TEMPLATE=CORRELATOR_STATUS
```

```
GLOBAL_CONSTANT=GC_LIST
```

```
CORRELATOR_TEMPLATE=TOK_LIST
```

```
CORRELATOR_NAME=TOK_LIST
```

This syntax includes the following parameters:

```
ALL_TEMPLATE=TOK_LIST
```

All correlator templates can be used by operators to edit the values of the parameters listed. *TOK\_LIST* can be any token identifier listed in [Token Identifiers in TOK\\_LIST](#) on page 247. However, any other condition specified in the Security file are *not* overridden by this statement.

```
ALL_TEMPLATE=CORRELATOR_STATUS
```

Operators can enable or disable the correlator to participate in correlation. If this condition is not specified, operators are not allowed to enable or disable correlators. By default, all correlators participate in correlation if it is already enabled.

```
GLOBAL_CONSTANT=GC_LIST
```

List of global constants whose values can be edited. *GC\_LIST* is a 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. `CORRELATOR_TEMPLATE` can be any correlator template name listed in [Template Names in CORRELATOR\\_TEMPLATE](#) on page 246. `TOK_LIST` can be any token identifier listed in [Token Identifiers in TOK\\_LIST](#) on page 247.

`CORRELATOR_NAME=TOK_LIST`

List of parameters for the specific correlator whose values can be edited. `CORRELATOR_NAME` is the name of the correlator template. `TOK_LIST` can be a token identifier listed in [Token Identifiers in TOK\\_LIST](#) on page 247.

### Template Names in CORRELATOR\_TEMPLATE

The `CORRELATOR_TEMPLATE` parameter includes the following correlator template names:

`ALL_TEMPLATE`

All correlator templates.

`ENHANCE`

Enhance correlator template.

`GLOBAL_CONSTANT`

Global constants.

`MULTI_SOURCE`

Multi-Source correlator template.

`RATE`

Rate correlator template.

`REPEATED`

Repeated correlator template.

`SUPPRESS`

Suppress correlator template.

`TRANSIENT`

Transient correlator template.

`USER_DEFINED`

User-Defined correlator template.

## Token Identifiers in TOK\_LIST

The *TOK\_LIST* parameter includes the following token identifiers:

ADVANCED\_FILTER

Advanced Filter.

ALARM\_SIGNATURE

Alarm Signature.

ALL\_PARAM

All parameters.

ALTER\_ALARM

Alter alarm parameters.

CLEAR\_ALM

Clear the alarms of the Transient correlator template.

COUNT

Count the number of alarms in the Rate correlator template.

CRT\_CALLBACK

Create Callback function parameters.

DESCRIPTION

Description of the correlator.

DIS\_CALLBACK

Discard Callback function section.

DISCARD

Discard alarm in the Rate correlator template.

DISCARD\_DUP

Discard duplicate in the Repeated correlator template.

DISCARD\_IMD

Discard immediately in the Repeated correlator template.

DISCARD\_ON\_SET

Discard alarms on set completion in the Multi-Source correlator template.

ENABLE\_THR

Enable a threshold in the Transient correlator template.

ENHANCE\_ALWAYS

Always enhance the alarm of the Enhance correlator template.

INPUT\_FUN

Input function in User-Defined correlation.

MESSAGE\_KEY

Message key.

NEW\_ALARM

New Alarm parameters.

OUTPUT\_FUN

Output function in User-Defined correlation.

PAR\_OTH CORR

Participate in other correlations in the Suppress correlator template.

SET

Wait for set completion in the Multi-Source correlator template.

THR\_CNT

Threshold count in the Transient correlator template.

THR\_WIN

Threshold window in the Transient correlator template.

VARIABLES

Variables.

WANT\_ORIGINAL

Want Original alarm in the Enhance correlator template.

WINDOW

Time period.



## Example of a Security File

A Security file might look like this:

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

This example indicates the following:

```
OV_Chassis_Cisco=NEW_ALARM
```

For the `OV_Chassis_Cisco` correlator, parameters defined for New Alarm creation can be edited.

```
USER_DEFINED=ALARM_SIGNATURE
```

For all User-Defined correlators, the values in the Alarm Signature section can be edited.

```
ALL_TEMPLATE=WINDOW
```

For all correlator templates other than the User-Defined template, as well as the `OV_Chassis_Cisco` correlator, the value for the window parameter can be edited.



To edit the value of the window parameter for the `USER_DEFINED` template or the `OV_Chassis_cisco` correlator, you must explicitly type the following:

```
OV_Chassis_cisco=NEW_ALARM,WINDOW
```

```
USER_DEFINED=ALARM_SIGNATURE,WINDOW
```

## Guidelines for Security Files

When creating Security files, follow these guidelines:

- **Conditions**

Specify each condition on a separate line.

- **Tokens**

Separate token parameters with commas.

- **Spaces**

Do not add blank space before or after the commas, which are used as separators for token identifiers.

- **Comments**

Precede all comments with a number sign (#).

- **Locations**

Always save the Security file as *<Correlator Store filename>.sec* in the directory where the Correlator Store is located.

- **Conditions**

For conditions in the Security file, follow this order of precedence:

- a Correlator name
- b Correlator template type
- c Condition for all templates

This order provides complete security and control for the Correlator Store.

- **Global constants**

Use the token identifier `GLOBAL_CONSTANT` to edit global constant values.

For example, if you want to give operators permission to edit the global constants `pi`, `timeout`, and `createtime`, type the following:

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

- **Correlator templates**

Provide appropriate token identifiers to make specific changes to values of attributes and variables in correlator templates. Specify all identifiers in uppercase. When adding token identifiers, follow the conventions listed in [Token Identifiers in TOK\\_LIST](#) on page 247.

## Creating Deploy Configuration Files

Operators are responsible for loading correlation logic into the HP Event Correlation Services (ECS) engine. For operator instructions, see [Chapter 10, Operator Mode in NNM](#).

Composer enables operators to load correlation logic into the ECS engine through the deploy feature. As the developer, you must maintain the Deploy Configuration file.

### Deploy Procedure

The deploy procedure invokes the `csdeploy` and `csmerge` scripts. These scripts merge the Correlator Store files, remove user descriptions from the merged Correlator Store, and then load the file into the ECS engine. If you want, you can execute the two scripts separately from the command prompt. For more information on how Correlator Stores are merged, see [Merging Correlator Store Files](#) on page 124.

The deploy procedure uses the Deploy Configuration file, which includes the following:

- Name of the Correlator Store file after the merge
- Path to the NameSpace file for the associated Correlator Store files
- Name of the log file to which the merge logs are written
- ECS engine instance to which the merged Correlator Store is loaded
- Logical name of the merged Correlator Store file

## Example of a Deploy Configuration Files

The Deploy Configuration file contains information required by the ECS engine at the time the Correlator Store files are loaded into the ECS engine. The Deploy Configuration file is an ASCII file.

A Deploy Configuration file might look like this:

```
#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.
#ENGINE_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
```

## Guidelines for Deploy Configuration Files

When creating a Deploy Configuration file, follow these guidelines:

- **Comments**

Precede all comments with a number sign (#). All text from the start of the number sign to the end of the current line is ignored by the system.

- **Locations**

Always specify file locations with the absolute path. Always enclose file locations within quotation marks. You can use environment variables when specifying file locations.

- **Blanks**

Do not add blank spaces before or after the equal sign (=).

- **Parameters**

Use the parameters listed in [Parameters for Deploy Configuration Files](#) on page 254.

## Parameters for Deploy Configuration Files

The Deploy Configuration file has the following parameters:

`SUPPORT_DEPLOY_ON_GUI`

Deploy the merged Correlator Store through the GUI.

`FINAL_CS_NAME`

Name of the merged Correlator Store file.

`NAMESPACE_FILE`

Name of the NameSpace file from which the Correlator Stores are gathered.

`MERGE_LOG_FILE`

Name of the log file to which the logs of the Correlator Store merge are written.

`ENGINE_INSTANCE`

ECS engine instance number for which the Correlator Store file is loaded.

`CS_LOGICAL_NAME`

Logical name of the merged Correlator Store.



Make sure the NameSpace file referenced in the Deploy Configuration file (see [Creating Deploy Configuration Files](#) on page 251) is the same file passed with the `-N` option when Composer is started (see the *ovcomposer* reference page). If the file names are different, you create one set of Correlator Stores, and then deploy a completely different set of Correlator Stores.

# Defining Operator Access

To provide access to Composer, you need the following for each operator:

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



To find out how to edit NameSpace and Security files in the NNM environment, see Chapter 10, “Correlation Composer for NNM,” on page 205.

To give operators access to Correlator Store files, complete the following tasks:

- Task 1: [Customizing the NameSpace File](#) on page 255
- Task 2: [Customizing the Security File](#) on page 256
- Task 3: [Customizing the Deploy Configuration File](#) on page 256



After creating the configuration files for your environment, make sure that their correct file names and locations are used when Composer is started. (For details, see the *ovcomposer* reference page.) If no files are specified, Composer uses the default configuration files.

## Customizing the NameSpace File

A default NameSpace file is available in the following location:

- *UNIX*  
`$OV_CONF/ecs/CIB`
- *Windows*  
`%OV_CONF%\ecs\CIB`

To override the specifications in the default NameSpace file, follow these steps:

- 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` or `%OV_CONF%\ecs\CIB`).

For guidelines, see [Guidelines for NameSpace Files](#) on page 244.

- 3 Save the file with a `.conf` extension.

## Customizing the Security File

When the Correlator Store file is saved the first time, a default Security file is created. This file is present in the directory where the Correlator Store is located.

To override the specifications in the default Security file, follow these steps:

- 1 List the token identifiers and the parameters that can be edited.

For guidelines, see [Guidelines for Security Files](#) on page 250.

- 2 Save the file as `<Correlator Store filename>.sec`.

Make sure that the file is stored in the same directory where the Correlator Store file is stored.



As a developer, you are responsible for providing the correct permissions for the file. Make sure the file cannot be overwritten or edited erroneously.

## Customizing the Deploy Configuration File

A default Deploy Configuration file is available in the following location:

- *UNIX*

`$OV_CONF/ecs/CIB`

- *UNIX*

`%OV_CONF%\ecs\CIB`

To override the specifications in the default Deploy Configuration file, follow these steps:

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

The newly created Namespace and Deploy Configuration files are bound together, based on the entry `NAMESPACE_FILE` in the Deploy Configuration file. Make sure you provide the correct Namespace file name in the Deploy Configuration file.



# Deploying the Correlator Store

As a developer, you deploy the Correlator Store to the ECS engine. Before doing so, 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 default Deploy Configuration file is located in the following directory:

- *UNIX*

`$OV_CONF/ecs/CIB/Devdeploy.conf`

- *Windows*

`%OV_CONF%\ecs\CIB\Devdeploy.conf`


Update this file to refer to the newly created `NameSpace` file.



Update the `Devdeploy.conf` file only if you do not want to disturb the existing configuration specifications in the `NameSpace.conf` file.

## Loading the Correlator Store File to the ECS Engine

To load the Correlator Store file into the ECS engine, follow these steps:

- 1 Make sure that all Correlator Stores have been saved and closed.
- 2 Do one of the following:
  - Click **Options**→**Deploy**.
  - Click the  **Deploy** icon.

## Viewing Errors in the Deploy Status Window

The Deploy Status window indicates one of the following:

- If the deployment is successful, the window indicates success.
- If an error occurred, the window indicates failure.

To view the details of an error, follow these steps:

- 1 In the Deploy Status window, click **Details**.
- 2 Click **OK** to close the window.

## Deploying Correlator Stores from the Command Prompt

You can deploy Correlator Stores from the command prompt using the `csdeploy.ovpl` script provided in the `$OV_BIN` or `%OV_BIN%` directory. The `csdeploy.ovpl` script references the Deploy Configuration file required by Composer. For details, see [Creating Deploy Configuration Files](#) on page 251.

To deploy the Correlator Store, type the following from the command prompt:

```
csdeploy.ovpl -p <Deploy Configuration file name>
```

This command contains the following parameter:

*<Deploy Configuration file name>*

Name of the Deploy Configuration file. If you do not specify a file name, the default Deploy Configuration file is selected:

- *UNIX*

`$OV_CONF/ecs/CIB/Devdeploy.conf`

- *Windows*

`%OV_CONF%\ecs\CIB\Devdeploy.conf`

To summarize the usage of `csdeploy`, run the **`csdeploy.ovpl -h`** command.

# 10 Operator Mode in NNM

This chapter explains how operators access and use HP Correlation Composer in HP Network Node Manager (NNM) environments:

- [Operator Access Rights](#) on page 259
- [Starting Composer in Operator Mode](#) on page 260
- [Locking Files](#) on page 261
- [Deploying Correlator Stores](#) on page 264

## Operator Access Rights

Correlation logic is created by the Correlator Store developer who provides access rights to operators. Operators have limited access on the Correlator Store files. These rights are governed by the information provided in the Security and NameSpace files.

In Composer, operators can do the following:

- **NameSpace file**

Access the files specified in the NameSpace file. This file is created and maintained by the developer. For details, see [Configuring Operator Profiles](#) on page 242. Only the files specified in the NameSpace file are visible to operators in the Composer NameSpace table.

- **Security file**

Edit values of parameters specified in the Security file. This file is created and maintained by the developer. For details, see [Configuring Operator Profiles](#) on page 242.

- **Correlator Stores**

Enable and disable correlators in Correlator Stores.

Operators *cannot* create new correlators and or Correlator Stores.

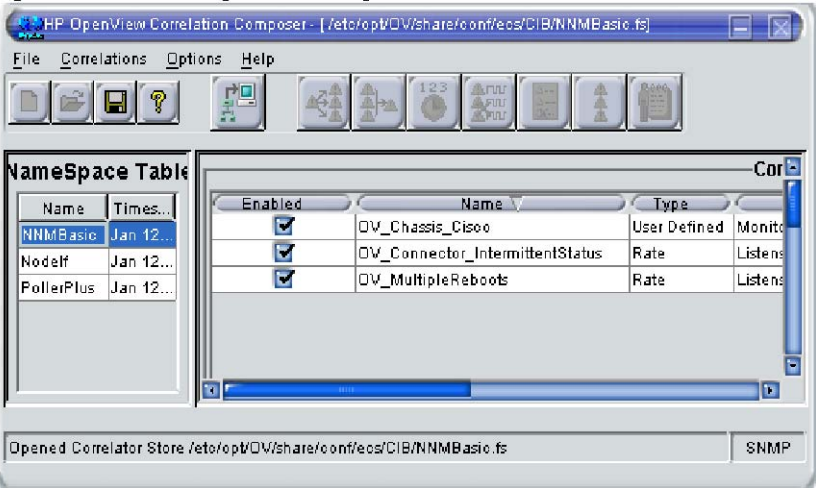
# Starting Composer in Operator Mode

To start Composer in Operator mode, type the following:

```
ovcomposer -m o
```

Composer opens in Operator mode with the list of Correlator Store files, as shown in [Figure 19](#). The last modified time of the Correlator Store displays in the Namespace table.

**Figure 19 Composer in Operator Mode**



# Locking Files

To avoid overwriting data when Correlator Stores are accessed concurrently by multiple users (developers or operators), Composer enables you to lock a file.

## File Locking Modes

File locking for Correlator Stores functions in the following modes:

- **Operator mode**

For details, see [Starting Composer in Operator Mode](#) on page 260.

- **Developer mode**

For details, see [Starting Composer in Developer Mode](#) on page 242.

- **Deploy script**

For details, see [Deploying Correlator Stores from the Command Prompt](#) on page 265.

- **Deploy procedure**

For details, see [Deploying Correlator Stores](#) on page 264.

- **Merge script**

For details, see [Merging Correlator Store Files](#) on page 124.

When you open a Correlator Store that is not already in use, a lock file is created. The lock file is named `<filename>.lock`, where `<filename>` is the name of the Correlator Store. Acquiring a lock provides total access to the Correlator Store file.

When you close the Correlator Store, the lock is removed.

## File Locking Failure

If you open a Correlator Store that is already in use, file locking fails:

- **Operator mode**  
Displays an error message and opens the file in read-only mode.
- **Developer mode**  
Displays an error message and aborts the file open action.
- **Deploy script**  
Displays an error message and aborts the deploy action.
- **Deploy procedure**  
Displays an error message and aborts the deploy action.
- **Merge script**  
Displays an error message and aborts the merge action.

## Recovering Correlator Stores

If a file action aborts while a Correlator Store is locked, you can recover the Correlator Store.



Recover Correlator Stores with caution. If multiple operators save Correlator Store files, it is possible to lose important data.

To recover a Correlator Store, follow these procedures:

- **Operator mode**

In Operator mode, highlight the locked Correlator Store and then click **Options→Forcefully Unlock**.



Although operators are allowed to make changes to the Correlator Store, they do not have exclusive access to this file.

- **Developer mode**

In Developer mode, remove the lock file manually from the directory where the Correlator Store is located.

- **Deploy script**

In Developer mode, remove the lock file manually from the directory where the Correlator Store is located.

- **Deploy procedure**

In Developer mode, highlight the locked Correlator Store and then select **Options→Forcefully Unlock**.

- **Merge script**


In Developer mode, remove the lock file manually from the directory where the Correlator Store is located.

# Deploying Correlator Stores

Operators can deploy a Correlator Store to the HP Event Correlation Services (ECS) engine.

## Loading the Correlator Store File to the ECS Engine

To load the Correlator Store file into the ECS engine, follow these steps:

- 1 Make sure that all Correlator Stores have been saved and closed.
- 2 Do one of the following:
  - Click **Options**→**Deploy**.
  - Click the  **Deploy** icon.

## Viewing Errors in the Deploy Status Window

The Deploy Status window indicates one of the following:

- If the deploy is successful, the window indicates success.
- If an error occurred, the window indicates failure.

To view the details of an error, follow these steps:

- 1 In the Deploy Status window, click **Details**.
- 2 Click **OK** to close the window.



## Deploying Correlator Stores from the Command Prompt

You can deploy Correlator Stores from the command prompt using the `csdeploy.ovpl` script provided in the `$OV_BIN` or `%OV_BIN%` directory. The `csdeploy.ovpl` script references the Deploy Configuration file required by Composer. For details, see [Creating Deploy Configuration Files](#) on page 251.

To deploy the Correlator Store, type the following from the command prompt:

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

This command contains the following parameter:

*<Deploy Configuration filename>*

Name of the Deploy Configuration file.

If you do not specify a file name, the default Deploy Configuration file is selected:

— *UNIX*

`$OV_CONF/ecs/CIB/Devdeploy.conf`

— *Windows*

`%OV_CONF%\ecs\CIB\Devdeploy.conf`

To summarize the usage of `csdeploy`, run the `csdeploy.ovpl -h` command.



# A Built-In Functions

HP Correlation Composer includes built-in functions and keys:

- **Functions**

Built-in functions help you log, retrieve, and manipulate event data.

- **Keys**

Values for the **store**, **retrieve**, **storeStr**, and **retrieveStr** functions are stored and retrieved against keys.

The keys are passed to the functions as parameters.

## Functions

HP Correlation Composer includes the following built-in functions:

- **add**

Sum of the values passed to the function.

- **bitand**

Integer value of a bitwise and operation between two arguments.

- **bitinv**

Integer value of a bitwise inversion of the argument.

- **bitor**

Integer value of a bitwise or operation between two arguments.

- **bitxor**

Integer value of a bitwise exclusive or operation between two arguments.

- **div**

Integer value of dividing one integer by another.

- `getByIndex`  
Element at the *index* position of a *list*.
- `getCounter`  
Counter values stored against the keys.
- `getHour`  
Integer representing the current hour.
- `getMinute`  
Integer representing the current minute.
- `getMonth`  
Integer representing the current month.
- `getTime`  
String representing the time (in seconds) since the epoch.
- `makeList`  
List of arguments that are passed to the function.
- `mod`  
Integer value of the remainder after dividing one integer by another.
- `mul`  
Integer that is the product of two multiplied values.
- `retrieve`  
Value that was stored previously.
- `retrieveStr`  
String that was stored previously.
- `setCounter`  
Incremented value stored under the keys.
- `store`  
Value to be stored, based on the keys for a given time period or until another call to store.
- `storeStr`  
String value to be stored, based on the keys for a specified time period.

- `sub`

Difference between two integers passed to the function.

## `add`

### **Syntax**

```
add int1 int2
```

### **Parameters**

*int1* and *int2* are integers.

### **Description**

Sum of values passed to the function.

### **Example**

**add 1 2** returns the value 3.

## `bitand`

### **Syntax**

```
bitand int1 int2
```

### **Parameters**

*int1* and *int2* are integers.

### **Description**

Integer value of a bitwise and operation between two arguments.

### **Examples**

**bitand 7 0** returns the value 0.

**bitand 7 1** returns the value 1.

## bitinv

### Syntax

`bitinv int`

### Parameters

*int* is an integer.

### Description

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`

### Parameters

*int1* and *int2* are integers.

### Description

Integer value of a bitwise or operation between 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`

### Parameters

*int1* and *int2* are integers.

### Description

Integer value of a bitwise exclusive or operation between 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`

### Parameters

*int1*

Integer dividend.

*int2*

Integer divisor.

### Description

Integer value of dividing the one integer by another.

### Example

**7 div 3** returns the integer 2.

## getByIndex

### Syntax

```
getByIndex list index failvalue
```

### Parameters

*list*

List of any data types.

*index*

Position from which the value is to be extracted.

*failvalue*

Value returned if the function fails.

### Description

Element at the *index* position of the *list* passed in. If *index* number of elements does not exist, the function returns the *failvalue*.

Typically, you use the `getByIndex` function to retrieve individual elements from the return value of the previous call to an external function.

### Examples

An external `getInterfaceDetails` function returns the `interfaceName` and `interface IP Address`. This return value is bound to a `details` variable.

To extract the IP address, you call the `getByIndex` function as follows:

```
getByIndex details 2 0
```

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

### See also

- [retrieve](#) on page 276
- [store](#) on page 280



## getCounter

### Syntax

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

### Parameters

*toInit*

Method in which the value is retrieved:

1

Returns the stored value and frees the storage memory occupied by this value.

0

Returns the stored value, but does not delete the storage space. Additional `retrieve` calls return the stored value.

*key1, key2, ...*

Keys based on which value is retrieved. To find out how keys function, see [Keys](#) on page 283.

### Description

Counter values stored against keys. The values are stored previously using the `setCounter` call with the same set of keys.

### Examples

The usage of the `getCounter` function is illustrated by the following example:

```
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 278

## getHour

### **Syntax**

```
getHour()
```

### **Description**

Integer representing the current hour. The value can be 0 through 23. All time is represented in Coordinated Universal Time (UTC).

## getMinute

### **Syntax**

```
getMinute()
```

### **Description**

Integer representing the current minute. The value can be 0 through 59. All time is represented in UTC.

## getMonth

### **Syntax**

```
getMonth()
```

### **Description**

Integer representing the current month. The value can be 1 through 12.

## getTime

### **Syntax**

```
getTime()
```

### **Description**

String representing the time in seconds since the epoch (January 1, 1970).

## makeList

### Syntax

`makeList arguments`

### Parameters

*arguments*

List of arguments that are passed to the function.

### Description

List of arguments passed to the function. Typically, you use the function as the input function, the output function, or both in user-defined correlators because the functions require a list as the return type.

### Examples

`makeList 10, 20, 30`

## mod

### Syntax

`int1 mod int2`

### Parameters

*int1* and *int2* are integers.

### Description

Integer value of the remainder after dividing one integer by another.

### 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
```

### Parameters

*int1* and *int2* are two integers.

### Description

Integer that is the product of two multiplied values.

### Examples

**mul 3 4** returns 12.

## retrieve

### Syntax

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

### Parameters

*toInit*

Method in which the values are retrieved:

1

Returns the stored value, and frees the storage memory occupied by this value.

0

Returns the stored value, but does not delete the storage space. Additional `retrieve` calls return the stored value.

*failvalue*

Value returned by the function if the `retrieve` function fails.

*key1, key2,...*

Keys based on which the value is retrieved. To find out how keys function, see [Keys](#) on page 283.

## Description

Value that was stored previously. You must call the values to be retrieved under the same keys in the same order.

## Examples

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

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

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

Retrieves the values associated with the keys `agent_addr` and `Constants.Type2_SP`, but does not delete the memory used for storage. All succeeding calls to `retrieve` return the stored value. If the `retrieve` function fails, the value returned is 0.

## See also

- [store](#) on page 280

## retrieveStr

### Syntax

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

### Parameters

*toInit*

Method in which the value is retrieved.

*failvalue*

Value returned by the function if the `retrieve` function fails.

*key1, key2,...*

Keys based on which value is retrieved. To find out how keys function, see [Keys](#) on page 283.

### Description

Value (as a string) stored previously by using the `storeStr` function, based on the same set of keys.

## Example

```
retrieveStr 1 0 agent_addr arrival_time
```

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

## See also

- [storeStr](#) on page 281

## setCounter

### Syntax

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

### Parameters

*toInit*

Method in which the value is set:

0

Increments the stored value by the amount specified in the *increment* value, but does not delete the storage space. Additional calls to retrieve return the stored value.

1

Re-initializes and frees the storage memory. The value passed is returned. The return value is the stored value.

2

Frees the associated memory if the resultant value after the operation (increment added to the stored value) is zero.

*increment*

Increment value.

*window*

Time period for which the value is stored.

*key1, key2, ...*

Keys against which the value is set. To find out how keys function, see [Keys](#) on page 283.

## Description

Incremented value stored under the keys. If no value has been previously stored, the value passed is stored. The value is stored for the time specified in the window. After that, the value is backed out.

For example, if the value in the counter before an operation is 10, and a `setCounter` operation is performed with an increment of 5 and a window of 3 seconds, the counter value is 15. After 3 seconds, the `setCounter` operation is reversed. In this example, the operation would result in 5 being decremented from the current counter value. If window is set to 0, the value is stored for 1 second. If window is set to -1, the value is stored until the keys are re-initialized.

## Examples

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

If the value stored previously is 5, the new value stored is  $6+5=11$ . If there was no value stored previously, the value stored is 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 273

## store

### Syntax

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

### Parameters

*value*

Value to be stored.

*window*

Time period for which the value is stored:

*n*

Time in seconds. The value is stored for *n* seconds.

-1

Value is stored forever.

*key1, key2, ...*

Keys based on which value is stored. To find out how keys function, see [Keys](#) on page 283.

### Description

Value to be stored, based on the keys for a given time period or until another call to store. There must be at least one key for the value being stored. Another call to store under the same keys overwrites the currently stored value.

### Examples

`store uuid agent_addr, Constants.Type2_SP`

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

### See also

- [retrieve](#) on page 276



## storeStr

### Syntax

`storestr toAppend separator value window key1, key2,...`

### Parameters

*toAppend*

Determines how the value is stored:

0

Appends the value to an existing value. Stores the value, based on the keys.

1

Stores the value, based on the keys. Any values stored previously are erased. Only the new value is stored.

*separator*

Field separator.

*value*

Value to be stored.

*window*

Time period for which the value is stored:

*n*

Time in seconds. The value is stored for *n* seconds.

-1

Value is stored forever.

*key1, key2...*

Keys based on which value is stored. To find out how keys function, see [Keys](#) on page 283.

### Description

String value to be stored, based on the keys for a specified time period.

## Examples

```
storeStr 0 ":" Hello 10 agent_addr arrival_time
```

Stores the string "Hello" for a period of 10 seconds.

```
storeStr 0 ":" World 5 agent_addr arrival_time
```

Stores World for 5 seconds.

A call to `retrieveStr` returns `Hello:World`. After 5 seconds, a call to `retrieveStr` returns `Hello`. After 10 seconds, a call to `retrieveStr` returns the *failvalue*.

## See also

- [retrieveStr](#) on page 277

## sub

### Syntax

```
sub int1 int2
```

### Parameters

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

### Description

Difference between two integers passed to the function.

### Example

**sub 20 10** returns the integer 10.

# Keys

This section describes how to use keys in the [store](#), [retrieve](#), [storeStr](#), and [retrieveStr](#) functions.

## Multiple Keys

For the `store`, `retrieve`, `storeStr`, and `retrieveStr` function, values are stored or retrieved against the keys passed to the function as parameters. The functions require at least one key to be passed. However, you can use multiple keys. If you use multiple keys, the function internally concatenates the values referred to by these keys and creates a single key.

For example, a key `x`, which holds a value `abc`, is equivalent to the set of keys `x`, `y`, `z` that hold the values `a`, `b`, `c` respectively. Make sure that the order of the keys is maintained. In this example, passing in keys `z`, `y`, `x` results in a final key value of `cba`, not `abc`.

## Unique Keys

The `store` and `retrieve` functions use a global hash table.



This table is a powerful mechanism for passing data between correlators. Using it incorrectly can result in correlators overwriting each other.

For example, `Correlator1` stores a value against a key whose value is `abc`, and `Correlator2` stores a value against a key whose value also evaluates to `abc`. In this situation, the value stored would be the last value stored. To ensure that correlators do not overwrite each other, choose unique keys. One way to ensure unique keys is to use the correlator name as part of the key.



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



---

## B Event Attributes

This appendix describes the valid attributes for all event types supported by HP Correlation Composer:

- [HPOM Event Attributes](#) on page 286
- [SNMP Event Attributes](#) on page 293

# HPOM Event Attributes

Table 17 describes the valid HP Operations Manager (HPOM) event attributes supported by Composer.

**Table 17 HPOM Event Attributes**

Message	Attribute Type	Description
ACTION_ACK	Boolean	Whether the message is acknowledged automatically on the HP Operations management server after the corresponding automatic action has finished successfully. Possible values are true or false.
ACTION_ANNOTATE	Boolean	Whether HP Operations creates “start” and “end” annotations for automatic actions. Possible values are true or false.
ACTION_CALL	String	Command to use as an automatic action for the HP Operations message. The default is an empty string. The maximum length is 2000 characters.
ACTION_NODE	String	System on which the automatic action runs. The default value is NODENAME. The maximum length is 254 characters.
ACTION_STATUS	Integer	Status of the automatic action that belongs to the current message. Possible values: 0 - ACTION_UNDEF Default. Action is undefined. 1 - ACTION_DEF Default if ACTION_CALL is defined. Action is defined. 2 - ACTION_STARTED Action is started. 3 - ACTION_FINISHED Action is finished.
APPLICATION	String	Application name to use for the HP Operations message. Default is an empty string. Maximum length is 32 characters.

**Table 17 HPOM Event Attributes**

Message	Attribute Type	Description
CREATION_TIME	Time	Time at which the message was created. The time is in UNIX format (seconds since the epoch). The default is the (local) time at which the message was created.
FORWARDED	Boolean	Read only. Whether message is forwarded in an environment configured with manager-to-manager forwarding. Possible values are <code>true</code> or <code>false</code> .
GROUP	String	HP Operations message group to use for the message. The default is an empty string. The maximum length is 32 characters.
INSTR_IF <sup>a</sup>	String	Name of the external instruction-text interface. This interface must be configured in HPOM. The default is an empty string. The maximum length is 36 characters.
INSTR_IF_TYPE <sup>a</sup>	Integer	Whether the internal HP Operations instruction-text interface or an external interface is used to display instructions for the message. Possible values: 0 - INSTR_NOT_SET Default. Instruction is not set. 1 - INSTR_FROM_OPC Instruction is stored in the HP Operations database. 2 - INSTR_FROM_OTHER Instruction is accessed by using an external instruction-text interface.
INSTR_PAR <sup>a</sup>	String	Parameters for the call to the external instruction-text interface. The default is an empty string. The maximum length is 254 characters.
MSG_LOG_ONLY	Boolean	Message is inserted immediately into the history message log when the message is received on the management server. The message is not sent to any operator. Operators can see the message only when using the HP Operations history message browser. Possible values are <code>true</code> or <code>false</code> .

**Table 17 HPOM Event Attributes**

Message	Attribute Type	Description
MSGID	String	Read only. Unique identifier of the message. Modified or newly created messages assume a null ID: 00000000-0000-0000-0000-000000000000.
MSGSRC	String	Read only. Source of the message. Examples: <ul style="list-style-type: none"> <li>• Name of the encapsulated log file if the message originated from log file encapsulation.</li> <li>• Interface name if the message was sent by using an instance of the Message Stream Interface (MSI).</li> </ul> The default is an empty string. There is no maximum length.
MSGSRC_TYPE	Integer	Read only. Source type of the message. Each source is represented in one bit. For example, a message that is generated by the log file encapsulator, and then modified at the Agent MSI, has bit or LOGFILE_SRC and AGTMSI_SRC set. Possible values: <ul style="list-style-type: none"> <li>1 - CONSOLE_SRC MPE/iX source.</li> <li>2 - OPCMSG_SRC OPC message source.</li> <li>4 - LOGFILE_SRC Log file source.</li> <li>8 - MONITOR_SRC Monitor source.</li> <li>16 - SNMPTRAP_SRC SNMP trap source.</li> <li>32 - SVMSI_SRC MSI on HP Operations management server.</li> <li>64 - AGTMSI_SRC MSI on HP Operations managed node.</li> <li>128 - LEGLINK_SRC Legacy Link interface.</li> <li>256 - SCHEDULE_SRC Schedule source.</li> </ul>



**Table 17 HPOM Event Attributes**

Message	Attribute Type	Description
MSGTEXT	String	Message text. The default is an empty string. There is no maximum length.
MSGTYPE	String	Message type used to group messages into subgroups (for example, to denote the occurrence of a specific problem). The default is an empty string. The maximum length is 36 ASCII characters with no blank spaces.
MSI_OUTPUT	Integer	<p>How messages are handled in the MSI. Each value represents one bit that can be processed with a bitor operation.</p> <p>Possible values:</p> <ul style="list-style-type: none"> <li>0 - SV_MSI_NO_OUTPUT Default. No message is sent to the Server MSI.</li> <li>1 - SV_MSI_DIVERT Divert the message to the Server MSI instead of sending it to the message manager.</li> <li>2 - SV_MSI_COPY Send the message directly to the message browser, and send a copy of the message to the Server MSI.</li> <li>0 - AGT_MSI_NO_OUTPUT Default. No message is sent to the Agent MSI.</li> <li>4 - AGT_MSI_DIVERT Divert the message to the Agent MSI instead of sending it to the message manager.</li> <li>8 - AGT_MSI_COPY Send the message directly to the message browser, and send a copy of the message to the Agent MSI.</li> </ul>
NODENAME	String	Name of the system on which the message is created. The message is handled by the HP Operations management server only if NODENAME is part of the HPOM managed environment (Node Bank). The default is the local node name. The maximum length is 254 characters.

**Table 17 HPOM Event Attributes**

Message	Attribute Type	Description
NOTIFICATION	Boolean	HP Operations messages are forwarded from the management server to the trouble ticket notification service interfaces if the appropriate notification interfaces are configured and active. Possible values are true or false.
OBJECT	String	Object name to use for the HP Operations message. The default is an empty string. The maximum length is 254 characters.
OPACTION_ACK	Boolean	Whether the message is acknowledged automatically on the HP Operations management server after the corresponding operator-initiated action has finished successfully. Possible values are true or false.
OPACTION_ANNOTATE	Boolean	Whether HPOM creates “start” and “end” annotations for the operator-initiated action. Possible values are true or false.
OPACTION_CALL	String	Command to use as an operator-initiated action for the HP Operations message. The default is an empty string. The maximum length is 2000 characters.
OPACTION_NODE	String	System on which the operator-initiated action should run. The default value is NODENAME. The maximum length is 254 characters.
OPACTION_STATUS	Integer	<p>Status of the operator-initiated action that belongs to the current message.</p> <p>Possible values:</p> <ul style="list-style-type: none"> <li>0 - ACTION_UNDEF Default. Action is undefined.</li> <li>1 - ACTION_DEF Default if OPACTION_CALL is defined. Action is defined.</li> <li>2 - ACTION_STARTED Action is started.</li> <li>3 - ACTION_FINISHED Action is finished.</li> <li>4 - ACTION_FAILED Action failed.</li> </ul>

**Table 17 HPOM Event Attributes**

Message	Attribute Type	Description
ORIGMSGTEXT	String	Original message text. This attribute enables you to set additional source information for a message. This source information is useful if the message text was reformatted, but the HP Operations operator needs to have access to the original text. The default is an empty string. There is no maximum length.
READ_ONLY	Boolean	Read only. Whether a message is forwarded as a “notification” in an environment configured with manager-to-manager forwarding. Possible values are true or false.
RECEIVE_TIME	Time	Read only. Time the message was received by the management server. The time is in UNIX format (seconds since the epoch). This value is set by the management server.
SERVICE_TIME	Time	Service time of the message.
SEVERITY	Integer	Severity of the message. Possible values: 4 - SEV_UNKNOWN 8 - SEV_NORMAL 16 - SEV_WARNING 32 - SEV_CRITICAL 64 - SEV_MINOR 128 - SEV_MAJOR
TIME_ZONE_DIFF	Time	Read only. Difference in seconds between Coordinated Universal Time (UTC) and local time at the time a message is created.

**Table 17 HPOM Event Attributes**

Message	Attribute Type	Description
TROUBLETICKET	Boolean	HP Operations messages are forwarded from the management server to the HP Operations trouble ticket interface if the interface is configured. Possible values are true or false.
TROUBLETICKET_ACK	Boolean	HP Operations management server acknowledges the message automatically if the message is forwarded to the trouble ticket system successfully. Possible values are true or false.
UNMATCHED	Boolean	Whether the message matches a condition. Possible values are true or false.

- a. HPOM for Windows, unlike HPOM for UNIX or Linux, does not support the instruction text interface. Using the HPOM event attributes INSTR\_IF, INSTR\_IF\_TYPE, and INSTR\_PAR in a correlator store leads to the following browser message: (MS387) An external Instruction Interface is not supported.

# SNMP Event Attributes

Table 18 describes the valid Simple Network Management Protocol (SNMP) event attributes supported by Composer.

**Table 18    SNMP Event Attributes**

Message	Attribute Type	Description
agent_addr	String in dot notation	Network address of the object that generated the trap in the form of an ECDL tuple.
enterprise	Object ID	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	Code that indicates the nature of the trap more specifically than the generic trap number. Specific trap numbers are defined by the owning enterprise. They are meaningful only in conjunction with the enterprise attribute.
time-stamp	Integer	Number of time ticks, in hundreths of a second, between the last initialization of the device and the generation of the trap. As a rule, this number is not required for correlation.
variable-bindings	String, integer	Additional information about the trap. The content of this field is dependent on the enterprise ID and specific trap values.



## C Pattern Matching

HP Correlation Composer includes a powerful text pattern-matching language that enables you to search logically for substrings and patterns. You can use the language to extract parts of a text string, assign the parts to tags, and then reuse the tags within the same scope.



The pattern-matching language provided by Composer is the same as that used in HP Operations Manager (HPOM).

Typically, you use pattern-matching to scan for a specific substring in the target string:

- **Positive matches**

To search for the substring `ERROR` anywhere in the target string, you would search for the following pattern:

```
"ERROR"
```

- **Negative matches**

To match text that does *not* contain the substring `WARNING`, you would search for the following pattern:

```
"<! [WARNING] >"
```

This string uses the NOT operator (`!`), angle brackets (`<>`) that enclose all operators, and brackets (`[]`) that isolate subpatterns.

- **Case sensitivity**

To control case sensitivity, you could use a separate argument to the `Match.make` function.

# Syntax of Pattern Matching

In pattern matching, you use the following expressions and operators:

- [Expression Delimiter \(\[ \]\)](#) on page 296
- [Operator Delimiter \(< >\)](#) on page 296
- [OR Operator \(|\)](#) on page 296
- [NOT Operator \(!\)](#) on page 297
- [Mask Operator \(\\)](#) on page 298

## Expression Delimiter ([ ])

You use brackets ([ ]) as delimiters to group expressions. Typically, you use bracketed expressions with the OR operator (|), the NOT operator (!), and when using subpatterns to assign strings to tags.

To increase performance, avoid brackets wherever possible. For example, in the pattern "ab[cd[ef]gh]", the brackets are unnecessary if you are searching for "abcdefgh".

## Operator Delimiter (< >)

You use angle brackets (<>) as delimiters for operators and expressions. You must enclose any expression that includes an operator (for example, a NOT operator) in angle brackets (for example, "<![WARNING]>").

## OR Operator (|)

You use a vertical bar (|) to separate alternate expressions with a logical OR. Expressions separated by the vertical bar match strings that contain either expression. For example, "[**ab**|**c**]**d**" would match the string "abd" and the string "cd".



## NOT Operator (!)

You use an exclamation point (!) to prefix expressions with a logical NOT operator. You must use the NOT operator with delimiting brackets. For example, "<![WARNING]>" would match all text that does *not* contain the string "WARNING".

You can also use the NOT operator with complex subpatterns:

```
"LN<*>: R< ![490| [501[a|b]]] >-<*>"
```

This complex pattern would enable you to generate a message for any line connection other than from repeaters 490, 501a, or 501b.

As a result, the complex pattern would match the following string:

```
"LN270: R300-427"
```

However, the complex pattern would not match the following string, which refers to repeater 501a:

```
"LN270: R501a-800"
```

If the subpattern that includes the NOT operator does *not* find a match, the NOT operator behaves like the <\*> expression: 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 number of characters, except 1, 2, or 3. The UNIX expression matches any *one* character, except 1, 2, or 3.

## Mask Operator (\)

You use a backslash (\) to mask characters that have a special meaning in pattern matching.

In pattern matching, the following special character have a special meaning:

[ ]

Brackets delimit expressions. For details, see [Expression Delimiter \(\[ \]\)](#) on page 296.

< >

Angle brackets delimit operators. For details, see [Operator Delimiter \(< >\)](#) on page 296.

|

Vertical bars separate expressions with logical OR operators. For details, see [OR Operator \(|\)](#) on page 296.

^

Carets delimit the beginning of lines. For details, see [Matching First or Last Characters](#) on page 299.

\$

Dollar signs delimit the ends of lines. For details, see [Matching First or Last Characters](#) on page 299.

If you want to mask the special function of one of these characters, you must preface it with a backslash (\) escape character (for example, \\$). A special character preceded by a backslash results in an expression that matches the special character itself.



An exception to this rule is the tab character, which is specified by entering `\t` into the pattern string.

The caret symbol (^) and the dollar sign (\$) have special meanings only when they are placed at the beginning or the end of a pattern, respectively. As a result, you do not need to mask them when using them within the pattern (that is, not as the first or last character of the pattern).

# Matching Expressions

You can define expressions to match the following:

- [Matching First or Last Characters](#) on page 299
- [Matching Multiple Characters](#) on page 300

## Matching First or Last Characters

You can use anchoring characters at the beginning or end of patterns:

- **Caret (^)**

If you use a caret as the first character of a 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".

- **Dollar sign (\$)**

If you use the dollar sign 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 you use a caret or dollar sign in any position other than the first or last character, it is considered an ordinary character without masking.

## Matching Multiple Characters

To match strings consisting of an arbitrary number of characters, you can use 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.

`<nS>`

Matches a string of  $n$  separators.

`<@>`

Matches any string that contains no separator characters. That is, it matches a sequence of one or more non-separators. You can use this expression to match words.

You can configure separator characters for each pattern. By default, separators are the space and the tab characters. You specify the separator string as the second element in the 3-tuple passed to the `Match.make` function.

## Matching Tags

In search patterns, you can use tags to identify parts 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 angle bracket.

You could define the following pattern:

```
^errno: <#.number> - <*.error_text>
```

This pattern would match the following string:

```
errno: 125 - device not in service
```

Also, it would assign 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.

## Assigning Substrings to Tags

In matching the pattern "<\*.tag1><\*.tag2>" against the string "abcdef", it is not immediately clear which substring of the input string is assigned to which tag. For example, it is possible to assign an empty string to tag1 and the whole input string to tag2. Likewise, it is possible to assign "a" to tag1 and "bcdef" to tag2.

The pattern matching algorithm always scans both the input line and the pattern definition (including alternative expressions) from left to right. The expression <\*> is assigned as few characters as possible. The expressions <#>, <@>, and <S> are assigned as many characters as possible. As a result, tag1 is assigned an empty string.

For example, you might want to match the following input string:

```
"this is error 100: big problem"
```

To match this input string, you would use a pattern such as the following:

```
error <#.errnumber>:<*.errtext>
```

This pattern includes the following assignments:

- "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 pattern, the colon (:) is used to delimit the expressions `<#>` and `<*>`.

Matching `<@.word><#.num>` against "abc123" assigns "abc12" to word and "3" to num, because 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.

As a result, the following two patterns would be treated the same way:

- `"this is number<#.num>"`
- `"<*>this is number<#.num><*>"`

## Assigning Subpatterns to Tags

In the same way that you can use a single operator (for example, `*` or `#`) to assign a string to a tag, you can use the operator to build a complex subpattern, composed of a number of operators with the following pattern:

`<[subpattern].tag>`

Examples:

`<[rack<#>.brd<#>].hware>`

The dot (.) between `rack<#>` and `brd<#>` matches a similar dot, but the dot between the close bracket (]) and `hware` is necessary syntax. This pattern would match a string such as "rack123.brd47" and assign the complete string to `hware`.

`<[Error|Warning].sev>`

Any line that contains either the word "Error" or "Warning" is assigned to the tag `sev`.

`<[Error[<#.n><*.msg>]].complete>`

Any line that contains the word "Error" has the error number assigned to the tag `n` and any further text assigned to `msg`. Both number and text are assigned to `complete`.

# Examples of Pattern Matching

You can use the ECS pattern matching language in many different ways.

Examples:

## **"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`, or `PANIC` at any place in the text, when `IgnoreCase` is specified.

## **"login|logout"**

Uses the OR operator (`|`) to recognize any message containing the keyword `login` or `logout`.

## **"^switch:<\*.msg> errno<\*><#.errnum>\$"**

Recognizes messages such as the following:

- `"switch: lost service errno : 6"`
- `"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`. The anchoring symbol (`$`) indicates that the digit `6` is matched only if it is at the end of the line.

## **"^errno[|=]<#.errnum> <\*.errtext>"**

Matches strings such as the following:

- `"errno 6 - no such device or address "`
- `"errno=12 not enough capacity. "`

Note the space before the OR operator (`|`). The expression within the brackets (`[]`) matches either this blank space or the equal sign (`=`). The blank space between `<#.errnum>` and `<*.errtext>` is used as a delimiter. Although not strictly required for assignments to the tags shown here, this blank space improves performance.

**"^system:<\*>:<\*.id>:"**

Matches a line delimited by colons such as the following:

"system:abc123:#103.79a:270295114730:"

It 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 the following:

"Warning: too many errors on circuit 473-186"

It assigns "too many errors" to `text`, and "473-186" to `circuit`.



## D Troubleshooting in NNM

This appendix explains how to troubleshoot HP Correlation Composer during run time in an HP Network Node Manager (NNM) environment:

- [Tracing Events](#) on page 306
- [Trace Tools](#) on page 310
- [Trace Configuration File](#) on page 311
- [Trace Messages](#) on page 313
- [Error Messages](#) on page 316



Before debugging correlators, it is helpful to understand the event flow through Composer. For an overview, see in [Event Flow](#) on page 24.

# Tracing Events

This section explains how to trace events in Composer:

- [Enabling Tracing in Composer](#) on page 306
- [Enabling Tracing in NNM](#) on page 307
- [Enabling Tracing for ECS](#) on page 308
- [Disabling Tracing for ECS](#) on page 308
- [Tracing the Flow of an Event](#) on page 309
- [Tracing the Actions of a Specific Correlator](#) on page 309
- [Tracing the Actions of a Correlator Event ID](#) on page 309

## Enabling Tracing in Composer

In Composer, you can enable tracing to follow program and data flow to debug problems. To perform tracing, you use the HP Cross-Platform Library (XPL).



Enabling tracing affects performance and trace file size. Enable tracing only when you need to troubleshoot problems.

To enable tracing, start Composer with the `-debug` option included:

**ovcomposer -debug**

## Enabling Tracing in NNM

To enable tracing in an NNM environment, follow these steps:

- 1 Make sure that the trace server is running.

To find out how to start the trace server, see the XPL documentation.

- 2 Edit the trace configuration file.

Examples:

- *HP-UX*

```
TCF Version 3.2
APP: "ECSComposer"
SINK: Socket "<server name>" ""
TRACE: "ECSTrace" "Event" Info Warn Error Developer Verbose
```

- *Windows*

```
TCF Version 3.2
APP: "ECSComposer"
SINK: Socket "<server name>" "node=<ip-address>";
TRACE: "ECSTrace" "Event" Info Warn Error Developer Verbose
```

In place of <server name>, include the name of the trace server. For trace file locations, see [Location of the Trace Configuration File](#) on page 311.

- 3 Associate the trace configuration file with the application that you are tracing by typing the following:

```
trccfg -server <server-name> <configuration filename>
```

- 4 Start the Trace Monitor utility by typing the following:

- *HP-UX or Linux*

```
run trcmon
```

- *Windows*

```
run tracemon
```

- 5 Start Composer.

## Enabling Tracing for ECS

To enable tracing for HP Event Correlation Services (ECS), follow these steps:

- 1 Enable tracing in NNM by typing the following command:

- *UNIX*

```
ecsmgr -fact_update Composer \  
$OV_CONTRIB/ecs/CO/CompTraceOn.fs
```

- *Windows*

```
ecsmgr -fact_update Composer  
%OV_CONTRIB%\ecs\CO\CompTraceOn.fs
```

 This command assumes that Composer is installed on the C drive.

- 2 Enable tracing for ECS by typing the following command:

- *NNM*

```
ecsmgr -i 1 -trace 65536  
pmdmgr -Secss\;T0xffffffff
```

- *HPOM management server*

```
ecsmgr -i 11 -trace 65536
```

- *HPOM agent*

```
ecsmgr -i 12 -trace 65536
```

## Disabling Tracing for ECS

To disable tracing for ECS, type the following command:

- *UNIX*

```
ecsmgr -fact_update Composer \  
$OV_CONTRIB/ecs/CO/CompTraceOff.fs
```

- *Windows*

```
ecsmgr -fact_update Composer  
%OV_CONTRIB%\ecs\CO\CompTraceOff.fs
```

## Tracing the Flow of an Event

To trace the flow of an event, type the following command:

```
grep "Composer" | grep <eventid>
```

In this command, *eventid* is the “if” of the event that needs to be traced.

## Tracing the Actions of a Specific Correlator

To trace the actions of a specific correlator, type the following command:

```
grep "Composer" | grep <correlatorname>
```

In this command, *correlatorname* is the name of the correlator that needs to be traced.

## Tracing the Actions of a Correlator Event ID

To trace the actions of a correlator on a given event ID, type the following command:

```
grep "Composer" | grep <correlatorname> | grep <eventid>
```

# Trace Tools

The HP Operations Tracing subsystem includes a variety of tools that help you control and monitor trace messages:

- **Trace server**

Process that provides an interface between trace-enabled applications and the tools you use to configure tracing in the applications or to monitor their trace output.

- **Trace configuration file**

File that contains the tracing specifications (for example, the application name being traced, the location of trace files, the kind of messages that you want to trace, and so on). Trace configuration files are ASCII text files that you can view or modify in a standard text editor.

- **Trace monitor**

Program (for example, `tracemon` or `trcmon`) that receives messages forwarded by the default trace server, and then displays them interactively or archives them to disk.

# Trace Configuration File

When an application initializes its trace configuration, it can get the initial trace configuration information from a variety of places. The configuration code searches for trace configuration information in a well-defined sequence of possible locations.

## Location of the Trace Configuration File

The trace configuration file is available at the following location:

- *UNIX (HP-UX, Linux, or Solaris)*  
`$OV_CONF/ecs/CO/OVCompTrc.tcf`
- *Windows*  
`%OV_CONF%\ecs\CO\OVCompTrc.tcf`

## Sample Trace Configuration File

A sample trace configuration file might look like this:

```
TCF Version 3.2
APP: "ECSComposer"
SINK: File "%OvInstallDir%\log\composer.trc" /
      "flush=1;maxfiles=10;maxsize=10;"
TRACE: "ECSTrace" "Event" Info Warn Error Developer Verbose /
Location Stack
```

If you choose static tracing, make sure that the trace configuration file is present in the current directory. When editing the trace configuration file, follow standards (for example, single spacing).

## Fields of the Trace Configuration File

In the trace configuration file, you must fill in the following fields:

- **TCF Version 3.2**

Version of the trace configuration file.

- **APP**

Application that is traced. In this case, Composer is traced.

- **SINK**

Location of the tracing component specifications for the trace file. This location can be overridden.

- **TRACE**

Two arguments. The first argument is the trace component name. The second argument is the trace category name. Enclose each argument in double quotes(" "). If you are using one of the standard categories in the code, it is mapped to a string value (which you supply).

## Viewing the Binary Trace Configuration File

The trace configuration file is in binary format. You can view it by using the Trace Monitor utility.

To view the binary file, start the Trace Monitor utility by typing the following:

- *HP-UX*

```
run trcmon
```

- *Windows*

```
run tracemon
```

On Windows, the Trace Monitor utility is a graphical user interface (GUI).



# Trace Messages

This section explains how to use trace messages to troubleshoot problems.

## Location of the Trace Message File

Trace message files are stored in the following directory:

- *UNIX (HP-UX, Solaris, or Linux)*

`$OV_LOG/pmd.log0`

- *Windows*

`%OV_LOG%\pmd.log0`



Enabling tracing affects performance and trace file size. Enable tracing only when you need to troubleshoot problems.

## Format of the Trace Message File

The trace file has the following format:

```
TRACE [interpreter]: Composer : <time stamp>.000000Z : <Event Id>
: <Correlator Name> : <Informative Message>
```

The trace file has six fields, separated by colons:

```
TRACE [interpreter]
```

Mandatory first field.

Composer

Mandatory second field.

```
<time stamp>.000000Z
```

Time stamp in the format `yyyymmddhhmmss`, followed by the time in milliseconds (always `000000Z`).

```
<Event Id>
```

ID of the event being processed.

*<Correlator Name>*

Name of the correlator that is being processed. If the trace message is part of the common processing of events, type **Common**.

*<Informative Message>*

Trace message itself.

## Reading the Trace Message File

After sending in events, you use the `grep` command for the string `Composer` to get trace messages. A trace message is printed when an event enters each of the phases described in [Tracing Events](#) on page 306.

When reading a trace message file, look for the following:

### 1 **Alarm Signature processing**

Alarm Signature processing begins and ends with the following strings:

- Alarm Signature processing starting
- Alarm Signature processing done

### 2 **Advanced Filter processing**

Advanced Filter processing begins and ends with the following strings:

- Advanced Filter processing starting
- Advanced Filter processing done

### 3 **Logic execution**

Logic execution begins and ends with the following strings:

- Input processing starting
- Input processing done

Look for the following two strings:

- Executing logic for the Correlator - starting
- Executing logic for the Correlator - done

The fifth field, `Correlator Name`, contains the name of the correlator that is executed. There may be a number of such pairs, depending on the number of correlators to be executed for the alarm.

## 4 Action processing

In action processing, look for the following strings:

- Decision after all the Correlators run

What happens to an event after all of the correlators have processed it.

- The variables are...

Variables defined in a correlator, which are periodically printed out as the correlator is being processed. Below the string is the set of variables that were defined for the correlator and the value currently bound to it. If the variable has not been evaluated yet, it is listed as Not Yet Evaluated. You may ignore variables that begin with an underscore (\_) because they are used for internal purposes only.

- Calling

Synchronous function that was called, and the arguments that are passed to it.

- Return value of the function

What the synchronous function returned.

- Asynchronously invoking function

Which asynchronous function was invoked, and which parameters are passed to it.

- Setting variable

Return values from asynchronous functions. This string indicates which variable is set for which correlator. Asynchronous functions are handled by a common code. As a result, the fifth field in the trace message is set to *Common*.

## 5 Event holding

If any correlators hold an event, the processing of the event continues after the specified period. In the meantime, other events may enter the system. To ensure that you are looking at the trace messages for the right event, check the `<Event Id>` field.

Event holding phases begin and end with the following strings:

- HOLD processing starting
- HOLD processing done.

There may be multiple such phases, depending on how many correlators requested a hold on the event.

# Error Messages

Before attempting to diagnose a problem, consider whether one of the following common situations may be the cause.

## Sample Error Messages

When Composer encounters problems during processing, it displays error traps, such as the following:

```
Trap-PDU {
enterprise {1 2 3 4},
agent-addr internet : "\x7F\x00\x00\x01",
generic-trap 6,
specific-trap 5000,
time-stamp 0,
variable-bindings {
{
name{1 2 3 4 5},
value simple: string : "Correlator Name is unctiionnotpresent :
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,
variable-bindings {
{
name{1 2 3 4 5},
value simple: string : "Asynchronous function call/s had errors
or times
out : Correlators affected are[annoNodeTimeOutInput]"
}
}
}
```

## Syntax for Error Messages

In Composer, error message use the following syntax:

Correlator Name is *Correlatorname:where:reason*

This error string contains the following parameters:

*Correlatorname*

Name of the correlator that caused the error.

*where*

At which point in the processing the error was encountered by a correlator:

— Post 'HOLD'

The correlator logic may hold an event for a specified period of time. For example, a User-Defined input function may hold an alarm for 10 seconds. After 10 seconds, the event exits from the hold, and the post-hold processing begins. In this example, the correlator involves 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*

Cause of the problem. For descriptions of selected error messages, see [Sample Error Messages](#) on page 316.

## Conventions for Error Messages

Error messages use the following conventions:

CORRELATORNAME

Name of the correlator that caused the problem.

FUNCTIONNAME

Name of the function that is called.

LIBRARYNAME

Name of the library that contains the user-supplied function.

ARGUMENTS

Parameters passed to the function

VARIABLENAME

Name of the variable being evaluated

For sample error messages, see [Sample Error Messages](#) on page 316.

## Sample Error Messages

When it encounters problems, Composer displays error messages, such as the following:

```
Function returned an Error - Error returned is Path = LIBRARYNAME  
- Function is FUNCTIONNAME: Function returned: ERROR STRING  
RETURNED BY THE FUNCTION
```

Called external function returned an error. The function was invoked as part of an evaluating variable in the correlator *CORRELATORNAME*.

```
'c' Function not found: Path = LIBRARYNAME - Function is  
FUNCTIONNAME
```

C function was not found in the specified library. Either the library name or the function name is incorrect.

```
Arguments to function is invalid Path = LIBRARYNAME - Function is  
FUNCTIONNAME: Function returned: PARAMETERS TO THE FUNCTION
```

Data types 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.

```
Library not found Path = LIBRARYNAME - Function is FUNCTIONNAME
```

Library path is not found. You could have entered an incorrect library name or the path specified is incorrect. Check the logs for more information.

```
Function timed out Path = LIBRARYNAME - Function is FUNCTIONNAME
```

Called function returned, but has not yet invoked, the callback function to indicate that processing is complete.

```
Function returned an Error - Error returned is Path = LIBRARYNAME  
- Function is FUNCTIONNAME: Function returned: ERROR STRING  
RETURNED BY THE FUNCTION
```

User-supplied function returned an error while processing.

```
Unable to evaluate variable - VARIABLENAME: No definition seems  
to exist for variable VARIABLENAME
```

Incorrect usage of automatic variable is most likely the cause of the problem. For example, using *OutputRetVal* as a parameter to the input function in User-Defined correlation would result in this error.

Lookup entry not found for key KEYNAME: Unable to evaluate  
VARIABLENAME

Key for which a Data Store lookup is being performed cannot be found. As a result, the variable cannot be evaluated.

Asynchronous function call/s had errors or timed out: Correlator is - CORRELATORNAME: variables are - VARIBALENAMES: Correlator is - CORRELATORNAME: Variables are -VARIABLENAME

Called function did not return values in the expected time. The most likely cause is that the function invoked did not send a return value.

Type mismatch while creating/altering the event. Attribute, Value pair is: (ATTRIBUTENAME, VALUE BEING ASSIGNED)

Variable type assigned to the attribute is incorrect (for example, assigning a string to the specific trap).

Variable Binding value not specified for index INDEXNUM

Value for the variable bindings with the index number *INDEX NUM* is not provided.

For conventions used in these error message, see [Conventions for Error Messages](#) on page 318.



# E Troubleshooting in HPOM

This appendix explains how to troubleshoot HP Correlation Composer during run time in an HP Operations Manager (HPOM) for UNIX environment.

In Composer, most problems result from minor problems with configuration or deployment.

When troubleshooting message correlation in the HP Event Correlation Services (ECS) engine at run time, you focus on a few common areas:

- Task 1: [Verifying Deployment](#) on page 322
- Task 2: [Message Logging](#) on page 326
- Task 3: [Statistics](#) on page 333
- Task 4: [Event Tracing](#) on page 335
- Task 5: [Error Logging](#) on page 344

In general, troubleshooting is similar for managed nodes (agents) and for the management server. Specific differences (typically, in file locations) are noted.



At run time, only the ECS engine runs. Composer fact stores configure the ECS circuit.

Before debugging correlators, it is helpful to understand the event flow through Composer. For an overview, see in [Event Flow](#) on page 24.

# Verifying Deployment

When you begin troubleshooting Composer—especially if this is the first time Composer has been used within your environment—make sure that the Correlation Composer template is configured and deployed correctly. While verifying the template, you see no effect on the message flow.

## Verifying the ECS Process

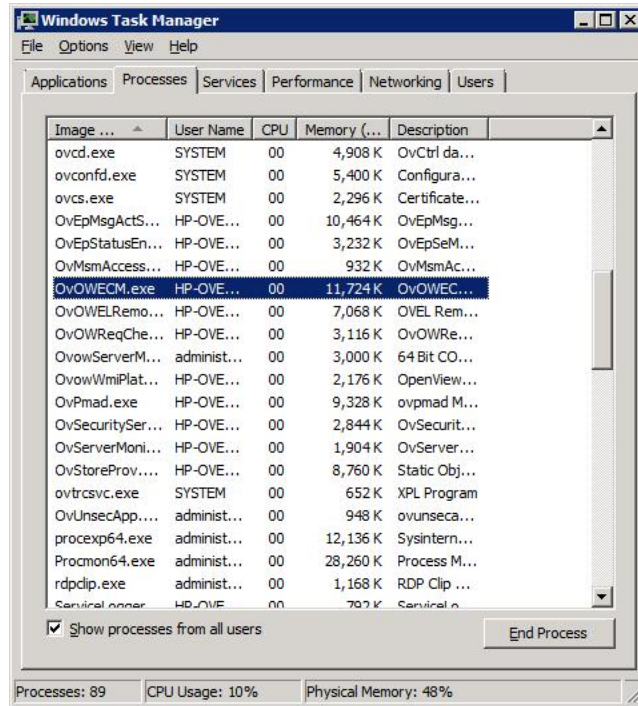
Composer is run from within the ECS process in HPOM. The ECS process does not run unless a Correlation Composer template is assigned and distributed to a given managed node or management server.

To verify the ECS process, follow these steps:

- 1 Do the following:
  - *Management server*
    - UNIX  
Run the **opcsv -status** command.
    - Windows  
Open the Task Manager.
  - *Managed node*  
Run the **opcagt -status** command.
- 2 Verify that the ECS process is running:
  - *Management server*  
Make sure the Event Correlation Manager process is running:
    - UNIX  
Verify that **opcecm** is running.

## — Windows

Verify that `OvOWECM.exe` is running.



- *Managed node*

Make sure the Event Correlation Agent (`opceca`) process is running.

- 3 If the ECS process is running, verify that Composer is currently running within ECS by running the `ecsmgr` command:

- *Management server*

```
ecsmgr -i 11 -info
```

- *Managed node*

```
ecsmgr -i 12 -info
```

The command is located in the following directory:

- *UNIX*

```
/opt/OV/bin/OpC/Utils/ecsmgr
```

- *Windows*

```
%OvInstallDir%\lbin\eaagt\ecsmgr
```

### Sample output:

```
Engine Status :
engine environment - Standalone Engine
engine instance - 11
engine version - ECS 3.33 (A.03.33)
time last started - Thu Aug 28 01:15:48 2008
engine trace mask - 0x0
engine log mask - 0xffffffff
maximum engine log size - 512 KBytes
maximum event log size - 512 KBytes
input event logging - off
default drill info logging - off
default drill event logging - off
automatic configuration saving - on

stream name - default
stream policy - output (unless discarded by a circuit)
stream event logging - off
stream policy event logging - off
stream drill info logging - off
stream drill event logging - off
circuits enabled in stream -
circuit ecs_comp

circuit name - ecs_comp
circuit date - Mon Dec 13 15:48:33 2004
circuit version - 0
circuit unique identifier - 5502542
time circuit load - Thu Aug 28 01:20:03 2008
circuit event logging - off
circuit state -
stream default - enabled
fact store name - ecs_comp
fact store date - 28 7 2008 17:9:57 36000000+0
fact store version - 1
time fact store load - Thu Aug 28 01:20:03 2008
```

The key item is the output indicating that the circuit `ecs_comp` is enabled:

```
circuits enabled in stream -
circuit ecs_comp
```

If these entries are *not* present in the (valid) output, the Enhance correlator deployed is *not* the Correlation Composer template.

- 4 Assign and deploy the Correlation Composer template if either of the following is true:
  - ECS process is *not* running.
  - ECS process is running, but Composer is *not* active in that process.

To find out how to assign and deploy the Correlation Composer policy correctly, see the following:

- [Configuring MSI in HPOM for UNIX or Linux](#) on page 145
- [Configuring MSI in HPOM 9.00 for Windows](#) on page 149

## Verifying the MSI Configuration

A common problem associated with the Correlation Composer template involves diverting messages to the HPOM for UNIX Message Stream Interface (MSI).

To verify the MSI configuration, follow these steps:

- 1 Verify that the messages to be processed by Composer display in the message log.

For details, see [Message Logging](#) on page 326.

- 2 If the messages to be processed by Composer do *not* appear in the input message log, reconfigure MSI.

When configuring MSI, follow these guidelines:

- *Management server*
  - MSI must be enabled on the server.
  - Either individual message source templates must divert messages to MSI, or all messages must be diverted to MSI.
- *Managed node*
  - MSI must be enabled on each agent where Composer is used.
  - Individual message source templates must divert messages to MSI.

For details, see the following:

- [Configuring MSI in HPOM for UNIX or Linux](#) on page 145
- [Configuring MSI in HPOM 9.00 for Windows](#) on page 149

## Verifying Composer File Deployment

The Composer configuration file that is deployed by using the `ovocomposer` command is named `ecs_comp.fs`. (For usage information, see [Merging and Deploying Correlator Store Files](#) on page 154.)

To verify Composer configuration file deployment, make sure the `ecs_comp.fs` file is in the following location:

- *Management server*
  - HPOM for UNIX or Linux  
`/var/opt/OV/shared/server/datafiles/policies/ec`
  - HPOM for Windows  
`%OvDataDir%\shared\server\datafiles\policies\ec`
- *Managed nodes*
  - HPOM for UNIX  
`/var/opt/OV/conf/eaagt`
  - HPOM for Windows  
`%OvDataDir%\conf\eaagt`

If the file is in this location, it was deployed successfully.

## Message Logging

If you do not find any problems in [Verifying Deployment](#) on page 322, verify the flow of messages. HPOM can log all events that are passed as Composer input and output.

You can use these event logs to verify the following:

- *Input*
  - Verify that messages flow to the ECS engine.
  - Verify that the expected messages are passed to the ECS engine.

- *Output*
  - Verify the suppression (not present in output) of messages.
  - Verify any modification of attributes in a message.

## Enabling Message Logging in HPOM for UNIX or Linux

To enable message logging for Composer in HPOM for UNIX or Linux, follow these steps:

- 1 Select and edit the **Correlation Composer** policy.
- 2 Click **Options**.
- 3 Select the **ECS log output** check box.
- 4 Select the **ECS log input** check box.
- 5 Assign and deploy the new version of the Correlation Composer policy:

- *Management server*

Select **Assign to Management Server** from the context menu. Then click **Deployment→Deploy Server Policies**.

Alternatively, use the following commands to deploy the Correlation Composer policy to the server:

```
opcnode -assign_pol pol_name='Correlation Composer'
pol_type=ec node_name='$MGMTSV'
net_type=NETWORK_NO_NODE

opcragt -distrib -force '$MGMTSV'
```

- *Managed nodes*

Select **Assign to Node / Node Group** from the context menu. Select the nodes to which you want to deploy the policy. Then click **Deployment→Deploy Configuration**.

Alternatively, use the following commands to assign and distribute the Composer EC Policy to the agents:

```
opcnode -assign_pol pol_name='Correlation Composer'
pol_type=ec node_name=<nodename> net_type=NETWORK_IP

opcragt -distrib <nodename>
```

After you enable message logging, all messages to and from Composer are logged in the following locations:

- **Input**
  - *Management server*  
`/var/opt/OV/log/OpC/mgmt_sv/ecevilg`
  - *Managed node*  
`/var/opt/OV/log/OpC/ecevilg`
- **Output**
  - *Management server*  
`/var/opt/OV/log/OpC/mgmt_sv/ecevolg`
  - *Managed node*  
`/var/opt/OV/log/OpC/ecevolg`

## Enabling Message Logging in HPOM 9.00 for Windows

To enable message logging for Composer in HPOM 9.00 for Windows, follow these steps:

- 1 In the console, click **Policy Management**→**Policies Grouped by Type**→**Event Correlation**.
- 2 Open the **Correlation Composer** policy.
- 3 Select the following check boxes:
  - **Log Messages that Leave the ECS Engine**
  - **Log Messages that Enter the ECS Engine**
- 4 Update the policy on the nodes on which you want logging to be enabled.

After you enable message logging, all messages to and from Composer are logged in the following locations:

- **Input**
  - *Management server*  
`%OvDataDir%\shared\log\inEcsEvt.log`
  - *Managed node*



%OvDataDir%\log\OpC\ecevilg

- **Output**

- *Management server*

%OvDataDir%\shared\log\ecs\_comp.log

- *Managed node*

%OvDataDir%\log\OpC\ecevolg

## Sample Message Log File

This section illustrates the format of the input and output message log files.

### Sample Input File

An input message log file might look like this:

```
> 03866cd8-74c9-71dd-06db-0f0276eb0000
> 1219904258 ; Thu Aug 28 00:17:38 2008
> hostname.your.com (IP) ; Normal ;
> Unmatched ; ; ITO: opcmsg(1|3)(1.0)
> ; Test ; Test
> ; :
> INSTR_NOT_SET: ;
> AA: ; ; ; Undef
> AA:
> OA: ; ;
> OA:
> NM Test1
> Node: \nMessage group: \nApplication: Test\nObject:
Test\nText: NM Test1\n
>
> 21600
>
>
>
> ;
>
% OpC_Msg:.:2
# 1219904260
+1042
```

## Sample Output File

In output message log files, the values of the input message are matched to HPOM message attributes.

An output message log file might look like this:

```
> [MSGID[;ORIGMSGID]]
> [CREATION_TIME] ; [CREATION_TIME (in ASCII)]
> [NODENAME] ([net_type: usually IP]) ; [SEVERITY] ;
[MSG_LOG_ONLY: "LOG_ONLY"|""]
> [UNMATCHED: "Matched"|"Suppressed"|"Unmatched"] ; [MSGTYPE] ;
"Console"|"OpC"|"Logfile"|"Monitor"|"SNMP"|"MSI": [MSGSRC]
> [GROUP] ; [APPLICATION] ; [OBJECT]
> [NOTIFICATION: "NOTIFICATION"|""] ; [TROUBLETICKET: "TT"|""] :
[TROUBLETICKET_ACK: "ACK"|""]
> [INSTR_IF_TYPE:
"INSTR_NOT_SET"|"INSTR_NOT_SET"|"INSTR_FROM_OTHER"] :
[INSTR_IF] ; [INSTR_PAR]
> AA: [AACTION_NODE] ; [AACTION_ACK: "ACK"|""] ;
[AACTION_ANNOTATE: "ANN"|""] ; [AACTION_STATUS:
Configuring Circuits for OVO
Logging Correlation Events in OVO
Chapter 3 103
"Undef"|"Def"|"Started"|"Finished"|"Failed"]
> AA: [AACTION_CALL]
> OA: [OACTION_NODE] ; [OACTION_ACK: "ACK"|""] ;
[OACTION_ANNOTATE: "ANN"|""] ; [OACTION_STATUS:
"Undef"|"Def"|"Started"|"Finished"|"Failed"]
> OA: [OACTION_CALL]
> [MSGTEXT]
> [ORIGMSGTEXT]
> [SERVICE_NAME]
> [TIME_ZONE_DIFF]
> [READ_ONLY: "READ_ONLY"|""]
> [FORWARDED: "FORWARDED"|""]
> [MSGKEY]
> [MSGKEY_RELATION_ICASE: "MKR_ICASE"|""] ;
[MSGKEY_RELATION_SEPS]
> [MSGKEY_RELATION]
% OpC_Msg:: (time difference between log time and CREATION_TIME)
# (log time in seconds since epoch)
+ (time difference to next message)
```

This sample output file contains the following notation:

[OPACTION\_ACK: "ACK" | ""]

Depending on the OPACTION\_ACK attribute, the log contains either ACK or nothing. The quotation marks are not present in the log file.

>

Beginning of a new line.

;

Separator for attributes on the same line.

XXX

Literal text that appears in the message.

[...]

Message attribute value. The brackets may enclose a list of literal values. The brackets are not present in the message.

"..."

Named value. The quotation marks are not present in the message itself. Empty quotation marks (" ") indicate that a missing value is acceptable.

|

Separator for alternative values.

(...)

Descriptive comment



The `ecsmgr -log_events` option is useful when troubleshooting in the NNM environment. For HPOM for UNIX environments, enable message logging as described in [Enabling Message Logging in HPOM for UNIX or Linux](#) on page 327.

# Statistics

ECS statistics provide you with a quick snapshot of messages that are currently being processed by Composer.

## Retrieving Statistics

To retrieve statistics, run the `-stats` command:

- *Management server*
  - UNIX

```
/opt/OV/bin/OpC/utlils/ecsmgr -i 11 -stats
```
  - Windows

```
"%OvInstallDir%\lbin\eaagt\ecsmgr" -i 11 -stats
```
- *Managed node*
  - UNIX

```
/opt/OV/bin/OpC/utlils/ecsmgr -i 12 -stats
```
  - Windows

```
"%OvInstallDir%\lbin\eaagt\ecsmgr" -i 12 -stats
```

## Sample Statistics

Output from the `-stats` command might look like this:

```
Engine Statistics -

input.inputFilters = [(), (), ()]
engineInstance = 12
currentTime = 20080828072139.000000Z
enginelog.errors = 0
enginelog.warnings = 1
enginelog.info = 3

Stream Statistics -

Stream "default" -

default.in.input = 0
default.in.new = 0
default.out.output = 0
default.out.discarded = 0
default.out.undecided = 0
default.out.errors = 0
default.original.output = 0
default.policy.num = 0

Circuit Statistics -

Circuit "ecs_comp" -

ecs_comp.in.input = 0
ecs_comp.in.new = 0
ecs_comp.out.output = 0
ecs_comp.out.discarded = 0
ecs_comp.out.undecided = 0
ecs_comp.out.errors = 0
ecs_comp.original.output = 0
```

These statistics tell you whether Composer is processing messages as expected. The details below Circuit "ecs\_comp" - provide the relevant information for Composer. (The Composer ECS circuit is named `ecs_comp`.)



These statistics are reset only when the ECS processes are restarted (normally on a server or agent restart).

The sample output includes the following statistics:

`ecs_comp.in.input = 0`

Number of messages input to Composer. This number indicates whether Composer is actually receiving any messages.

`ecs_comp.in.new = 0`

Number of new messages generated by Composer. An Enhance correlator that alters a messages creates a new message.

`ecs_comp.out.output = 0`

Number of messages output. This number includes the original messages output, as well as the newly created messages.

`ecs_comp.out.discarded = 0`

Number of messages discarded by Composer. This number includes messages in which an input message has been altered, and the newly created message is output, but the actual original message is discarded.

`ecs_comp.out.undecided = 0`

Number of messages currently held by Composer until a decision is made whether to discard or output. For example, Multi-Source and Transient correlators must often wait for appropriate matching messages. During this time, the number of the messages held is in this undecided count.

`ecs_comp.out.errors = 0`

Number of errors. If Composer cannot a process a message, it increments the errors count. If this count increases, see [Event Tracing](#) on page 335.

`ecs_comp.original.output = 0`

Number of original messages that were input to and output from Composer.

## Event Tracing

Tracing Composer events in run time provides you with troubleshooting information about the processes in your configuration.



Enabling event tracing affects performance and trace file size. Disable event tracing when you are not actively using it for troubleshooting.



## Enabling Tracing

To enable event tracing on the system, follow these steps:

### 1 Enable event tracing in Composer.

Run the following command:

- *Management server*

- UNIX

```
ecsmgr -i 11 -fact_update ecs_comp \  
$OV_CONTRIB/ecs/CO/CompTraceOn.fs
```

- Windows

```
ecsmgr -i 11 -fact_update ecs_comp  
%OvInstallDir%\ecs\CO\CompTraceOn.fs
```



On Windows, you cannot use a back slash (\) to escape new lines. You must type the entire command on one line at the command prompt.

- *Managed node*

- UNIX

```
ecsmgr -i 12 -fact_update ecs_comp \  
$OV_CONTRIB/ecs/CO/CompTraceOn.fs
```

- Windows

```
ecsmgr -i 12 -fact_update ecs_comp  
%OvInstallDir%\ecs\CO\CompTraceOn.fs
```

### 2 Enable event tracing in ECS.

In ECS, run the following command:

- *Management server*

```
ecsmgr -i 11 -trace 65536
```

- *Managed node*

```
ecsmgr -i 12 -trace 65536
```

## Disabling Tracing

To disable event tracing, follow these steps:

### 1 Disable event tracing in Composer.

In Composer, run the following command:

- *Management server*

- UNIX

```
ecsmgr -i 11 -fact_update ecs_comp \  
$OV_CONTRIB/ecs/CO/CompTraceOff.fs
```

- Windows

```
ecsmgr -i 11 -fact_update ecs_comp  
%OvInstallDir%\contrib\ecs\CO\CompTraceOff.fs
```



On Windows, you cannot use a back slash (\) to escape new lines. You must type the entire command on one line at the command prompt.

- *Managed node*

- UNIX

```
ecsmgr -i 12 -fact_update ecs_comp \  
$OV_CONTRIB/ecs/CO/CompTraceOff.fs
```

- Windows

```
ecsmgr -i 12 -fact_update ecs_comp  
%OvInstallDir%\contrib\ecs\CO\CompTraceOff.fs
```

### 2 Disable event tracing in ECS.

In ECS, run the following command:

- *Management server*

```
ecsmgr -i 11 -trace 0
```

- *Managed node*

```
ecsmgr -i 12 -trace 0
```

## Location of the Trace Message File

The trace message file is stored in the following location:

- *Management server*
  - UNIX  
`/var/opt/OV/log/OpC/mgmt_sv/ecengtr`
  - Windows  
`%OvDataDir%\log\OpC\mgmt_sv\ecengtr`
- *Managed node*
  - UNIX  
`/var/opt/OV/log/OpC/ecengtr`
  - Windows  
`%OvDataDir%\log\OpC\ecengtr`

## Format of the Trace Message File

The trace message file has the following format:

```
TRACE [interpreter]: Composer : <time stamp>.000000Z : <Event Id>  
: <CorrelatorName> : <Informative Message>
```

The trace message file has six fields, separated by colons:

TRACE [interpreter]

First field that is always set and that is always the same.

Composer

Second field that is always set and that is always the same.

<time stamp>.000000Z

Time stamp in the format `yyyymmddhhmmss`, followed by the time in milliseconds (always `000000Z`).

<Event Id>

ID of the event being processed.

*<Correlator Name>*

Name of the correlator that is being processed. If the trace message is part of the common processing of events, type **Common**.

*<Informative Message>*

Trace message itself.

## Sample Trace Entries

Entries in a Composer trace message file might look like this:

```
TRACE [interpreter]: Composer : 20080828065408.000000Z :  
"eventId(0:3744)" : Common : Advanced signature processing  
starting  
  
Common : Advanced signature processing starting  
Common : Alarm signature processing starting  
Test1 : "APPLICATION" "=" "Test"  
Test1 : "Test" "=" "Test"  
Test1 : The result of above comparison is : true  
Test1 : Incoming Alarm passed Alarm signature for this correlator  
Common : Alarm signature processing done  
Test1 : Executing logic for the Correlator - starting  
Test1 : In Enrich correlation input Processing (Correlator Test1)  
:The variables are:  
Test1 : __isRuleValid = true  
Test1 : __corClass = 5  
Test1 : newVal = "Hello"  
Test1 : The Correlator has decided the following - :Event will be  
altered.  
Test1 : Executing logic for the Correlator - done  
Common : Decision after all the Correlators ran: The event will  
be Altered.
```

The first (TRACE) line shows a complete entry. The subsequent (Common, Test 1) lines shows the details of the trace entries.

## Reading the Trace Message File

After Composer processes events, it logs entries to the trace message file.

When reading a trace message file, look for the following:

### 1 **Alarm Signature processing**

Alarm Signature processing begins and ends with the following strings:

- Alarm Signature processing starting
- Alarm Signature processing done

### 2 **Advanced Filter processing**

If the filter criteria of at least one correlator was met, Advanced Filter processing begins and ends with the following strings:

- Advanced Filter processing starting
- Advanced Filter processing done

### 3 **Logic execution**

Logic execution begins and ends with the following strings:

- Input processing starting
- Input processing done

Look for the following two strings:

- Executing logic for the Correlator - starting
- Executing logic for the Correlator - done

The fifth field, Correlator Name, contains the name of the correlator that is executed. There may be a number of such pairs, depending on the number of correlators to be executed for the alarm.

#### 4 Action processing

In action processing, look for the following strings:

- Decision after all the Correlators run  
What happens to an event after all of the correlators have processed it.
- The variables are...  
Variables defined in a correlator, which are periodically printed out as the correlator is being processed. Below the string is the set of variables that were defined for the correlator and the value currently bound to it. If the variable has not been evaluated yet, it is listed as Not Yet Evaluated. You may ignore variables that begin with an underscore (\_) because they are for internal use only.
- Calling  
Synchronous function that was called and the arguments that are passed to it.
- Return value of the function  
What the synchronous function returned.
- Asynchronously invoking function  
Which asynchronous function was invoked, and which parameters are passed to it.
- Setting variable  
Return values from asynchronous functions. This string indicates which variable is set for which correlator. Asynchronous functions are handled by a common code. As a result, the fifth field in the trace message is set to *Common*.

## 5 Event holding

If any correlators hold an event, the processing of the event continues after the specified period. In the meantime, other events may enter the system. To ensure that you are looking at the trace messages for the right event, check the `<Event Id>` field.

Event holding phases begin and end with the following strings:

- HOLD processing starting
- HOLD processing done

There may be multiple such phases, depending on how many correlators requested a hold on the event.

## Tracing the Flow of a Message

To trace the flow of a message, type the following command:

- *UNIX*  
**grep "Composer" | grep <eventid>**
- *Windows*  
**find "Composer" | find <eventid>**

In this command, *eventid* is the identifier of the event that needs to be traced.



This is an internal ECS event ID, not an HPOM for UNIX message ID (MSGID) attribute.

## Tracing the Actions of a Specific Correlator

To trace the actions of a specific correlator, type the following command:

- *UNIX*  
**grep "Composer" | grep <correlatorname>**
- *Windows*  
**find "Composer" | find <correlatorname>**

In this command, *correlatorname* is the name of the correlator that needs to be traced.

## Tracing the Actions of a Correlator Event ID

To trace the actions of a correlator on a given *eventid*, type the following command:

- *UNIX*  
**grep "Composer" | grep <correlatorname> | grep <eventid>**
- *Windows*  
**find "Composer" | find <correlatorname> | find <eventid>**

## Error Logging

The ECS engine in which Composer runs logs any specific errors to the engine log file. Look at this file if other troubleshooting tasks do not help you find the source of a problem.

### Format of the Error Log

The engine error file is formatted as follows:

`LOG [<Internal ECS filename>]: <Informative Message>`

Only *<Informative Message>* is used for troubleshooting.

### Sample Error Log Entries

Error log entries might look like this:

```
LOG [../perl_interp.c@227]: Perl Path /var/opt/OV/conf/OpC/
mgmt_sv/perlfile.pl
LOG [../perl_interp.c@345]: Error: while accessing perl file /
var/opt/OV/conf/OpC/mgmt_sv/perlfile.pl
LOG [../perl_interp.c@227]: Perl Path /opt/OV/contrib/ecs/
external/perl/getData.p
```



---

# F Error Messages

This appendix lists the error messages you might encounter when creating or deploying Correlation Stores in HP Correlation Composer:

- [Creation Errors](#) on page 346
- [Deployment Errors](#) on page 352

For each error message, it describes the cause of the problem as well as the action, if any, you should take to correct the problem.

# Creation Errors

When creating Correlator Store files, you might encounter the following error messages:

- [Alarm Name Is In Use, Cannot Delete](#) on page 347
- [Correlator Name Is Invalid](#) on page 347
- [Duplicate Alarm Name](#) on page 347
- [Duplicate Variable Name](#) on page 347
- [Invalid Syntax](#) on page 348
- [Look and Feel Not Supported](#) on page 348
- [Minutes in Window Period Cannot Be Greater Than 60](#) on page 348
- [No Blank Entry Allowed](#) on page 348
- [Seconds in Window Period Cannot Be Greater Than 60](#) on page 349
- [Threshold Count Should Be Integer Only](#) on page 349
- [Unknown Event Type](#) on page 349
- [Unspecified Correlator Name](#) on page 349
- [Unspecified Function Name](#) on page 350
- [Unspecified Threshold Count](#) on page 350
- [Unspecified Threshold Window](#) on page 350
- [Unspecified Window Period](#) on page 350
- [Variable in Use, Cannot Delete](#) on page 351
- [Variable Name Cannot Be Null](#) on page 351
- [Window Period Should Be Integer Only](#) on page 351

## Alarm Name Is In Use, Cannot Delete

### **Problem**

In Multi-Source correlation, you are trying to delete an existing alarm whose attributes are being used to create a new alarm.

### **Solution**

Change the New Alarm specification and delete the alarm.

## Correlator Name Is Invalid

### **Problem**

You have entered a blank space or special character in the Correlator name text field.

### **Solution**

Rename the correlator without any blank spaces or special characters.

## Duplicate Alarm Name

### **Problem**

Specified correlator name already exists.

### **Solution**

Enter a different name for the correlator.

## Duplicate Variable Name

### **Problem**

Specified variable name already exists.

### **Solution**

Provide a new name for the variable.

## Invalid Syntax

### **Problem**

You have entered some invalid characters (for example, consecutive blank spaces, commas, or mismatched quotation marks).

### **Solution**

Enter values without consecutive blank spaces, commas, or mismatched quotation marks.

## Look and Feel Not Supported

### **Problem**

You have tried to change the look and feel to “Windows” on a UNIX machine.

### **Solution**

Re-enter the Threshold Count in numeric form only.

## Minutes in Window Period Cannot Be Greater Than 60

### **Problem**

You have tried to enter a minutes value greater than 60.

### **Solution**

Convert minutes to hours, and then enter the value in the Window Period field.

## No Blank Entry Allowed

### **Problem**

You have left an entry blank.

### **Solution**

Enter valid values in the blank cell.

## Seconds in Window Period Cannot Be Greater Than 60

### **Problem**

You have entered a seconds value greater than 60.

### **Solution**

Convert the seconds value to minutes, and then enter the value in the Window Period field.

## Threshold Count Should Be Integer Only

### **Problem**

You have tried to enter special characters or unsupported alphabetical characters in the Threshold Count window.

### **Solution**

Re-enter the Threshold Count in numeric form only.

## Unknown Event Type

### **Problem**

You entered an invalid event type while starting the Orchestrator from the command line.

### **Solution**

Enter a valid event type: SNMP or OpC.

## Unspecified Correlator Name

### **Problem**

You have not entered a name for the correlator.

### **Solution**

Enter a name for the correlator.

## Unspecified Function Name

### **Problem**

You have not entered the name for the function while defining a callback function.

### **Solution**

Enter the name of the function.

## Unspecified Threshold Count

### **Problem**

In Transient correlation, you have tried to correlate without specifying the count of the `Failure` and `Clear` pairs after checking the `Enable Threshold` checkbox.

### **Solution**

Enter the threshold count.

## Unspecified Threshold Window

### **Problem**

In Transient correlation, you have tried to correlate without specifying the threshold window in which you want to monitor the correlation.

### **Solution**

Enter the threshold window in which you want to monitor the correlation.

## Unspecified Window Period

### **Problem**

You have not entered the window period in which the correlation has to be maintained.

### **Solution**

Enter the window period in which the correlation must be entered in the `Time Period` window.

## Variable in Use, Cannot Delete

### **Problem**

You have tried to delete a variable that is used to define other variables.

### **Solution**

Change the condition rule for the variable, and then delete the variable.

## Variable Name Cannot Be Null

### **Problem**

You have not entered a name for the variable.

### **Solution**

Enter a name for the variable.

## Variable Name in Use, Cannot Rename

### **Problem**

You have tried to rename a variable that is used when defining other variables.

### **Solution**

Change the condition rule for the variable, and then rename the variable.

## Window Period Should Be Integer Only

### **Problem**

You have tried to enter special characters or unsupported alphabetical characters in the Time Period window.

### **Solution**

Re-enter the window period in numeric form only.

# Deployment Errors

When merging and deploying Correlator Store files, you might encounter the following error messages:

- [Cannot Load the Correlator Store into the ECS Engine](#) on page 352
- [Merge Failed: Cannot Execute the csmerge Script](#) on page 352
- [Merge Failed: Cannot Open File](#) on page 353
- [Merge Failed: Correlator Stores Are of Different Event Type](#) on page 353
- [Merge Failed: Correlator Stores Have Different C Libraries](#) on page 353
- [Merge Failed: Correlator Stores Have Different Perl Files](#) on page 353
- [Merge Failed: Destination File Already Exists](#) on page 354

## Cannot Load the Correlator Store into the ECS Engine

### **Problem**

The Correlator Store cannot be loaded into the ECS engine.

### **Solution**

Verify that the ECS engine is running. If it is already running, contact the administrator.

## Merge Failed: Cannot Execute the csmerge Script

### **Problem**

The `csmerge` script (invoked internally at the time of deployment) could not be executed.

### **Solution**

None.



## Merge Failed: Cannot Open File

### **Problem**

One of the Correlator Store files specified in the NameSpace file cannot be opened.

### **Solution**

Check the permissions for the file. Change the permissions for the file to ensure that the file is visible.

## Merge Failed: Correlator Stores Are of Different Event Type

### **Problem**

The `csmerge` file is trying to merge Correlator Store files created for different event types.

### **Solution**

None. Correlator files created for different event types cannot be merged.

## Merge Failed: Correlator Stores Have Different C Libraries

### **Problem**

The C libraries are different in the Correlator Stores.

### **Solution**

Reference only one C library when specifying a C library name.

## Merge Failed: Correlator Stores Have Different Perl Files

### **Problem**

The Perl files are different in the Correlator Stores.

### **Solution**

Include all Perl files in one main Perl file, and then reference the main Perl file when specifying the Perl file name. For details, see [Support for Multiple Perl Files](#) on page 115.

## Merge Failed: Destination File Already Exists

### **Problem**

A Correlator Store file with the same name already exists.

### **Solution**

Provide a different name to the merged Correlator Store.

# Glossary

## **Advanced Filter**

Condition used to further filter alarms that have entered a rule. Typically, you use this condition to define filters based on external factors (for example, state and topology).

## **Alarm Signature**

Set of event attributes (Attribute, Operator, and Value) on which the first level of filtering is based. Further processing takes place when an event matches all attributes in the alarm set.

## **attribute**

Set of named value pairs, in which the name is an attribute.

## **callback function**

User-defined function invoked when a new alarm is created or when an existing alarm is discarded. When applied to an event, a correlation rule may discard an event or generate a new correlation. You can invoke user-defined functions to perform user-specific functions (for example, logging events).

## **combine**

Operator used to combine variables to form a new variable.

## **Composer**

HP Correlation Composer. Combination of a packaged ECS circuit and the graphical user interface (GUI) used to add parameters and define correlation rules for event correlation.

## **constant**

Values used for reference when defining a correlation rule. The variable name is bound to the value specified in the value field.

**correlation**

Processing an event stream to improve its value (for example, by making it smaller and by improving its information content). This processing is performed on the basis of relationships between events.

**correlator**

Uniquely identified unit of correlation logic that is applied to an event or a set of events.

**Correlator Store**

Set of correlators that define a specific correlation requirement.

**ECS**

HP Event Correlation Services. Service that identifies and highlights changes in the state of a network (by suppressing and correlating event storms), and then passes on or generates events that have a greater significance or a higher value.

**Enhance correlation**

Correlation type that enables event attributes of an event to be added, deleted, or changed after correlation. This type of correlation changes the information content of an event.

**Event Correlation Services**

See [ECS](#).

**extract**

Retrieving substrings within an event attribute.

**function**

Variable type whose value can be associated with the name of a variable. The return value of a user-defined function can be bound to a variable name.

**global constants**

Values bound to names when defining correlation rules.

**lookup operator**

Operator used to query values from a Data Store and bind them to variables. The return type of this operator is always the same type as the value in the Data Store. However, if you specify more than one parameter, the return value is a combined string.

**message key**

Key identifying the instance of the rule under which an alarm is correlated after it has passed the Alarm Signature and Advanced Filter conditions.

**Multi-Source correlation**

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

**parameters**

Measurable factors that change the default behavior of a basic rule type. The functionality of these factors can vary across the different correlation types.

**Rate correlation**

Correlation that measures the number of events occurring in a defined window of time (threshold time), and then outputs an event with more information content. The rate is maintained for a particular category of events. An arrival rate that exceeds the threshold indicates a serious problem. For this reason, a new alarm is created and forwarded.

**Repeated correlation**

Correlation used to eliminate similar alarms. This correlation passes on only those alarms that contain useful information. As a result, operators do not need to receive alarms of the same kind during a defined window of time.

**Suppress correlation**

Correlation used when a specific category of alarms must be discarded. These alarms are identified by the specified Alarm Signature.

**Transient correlation**

Correlation used to detect transient alarms. Alarms that come from the same network element, and that have the same probable cause and specific problem within a specified time window, are considered transient. You can define a period of time in which this transience must occur. To do so, you check for the number of transient alarms occurring in a threshold of time. If the transient suppression exceeds the threshold, a new alarm is generated and forwarded to operators. The new alarm alerts operators to threshold breach, and indicates the number of transients suppressed in the window.

**User-Defined correlation**

Correlation used to implement a requirement when none of the other correlation models, either by itself or in a combination, can meet the correlation requirement.

**Variables**

Names given to values used to define a correlation rule.