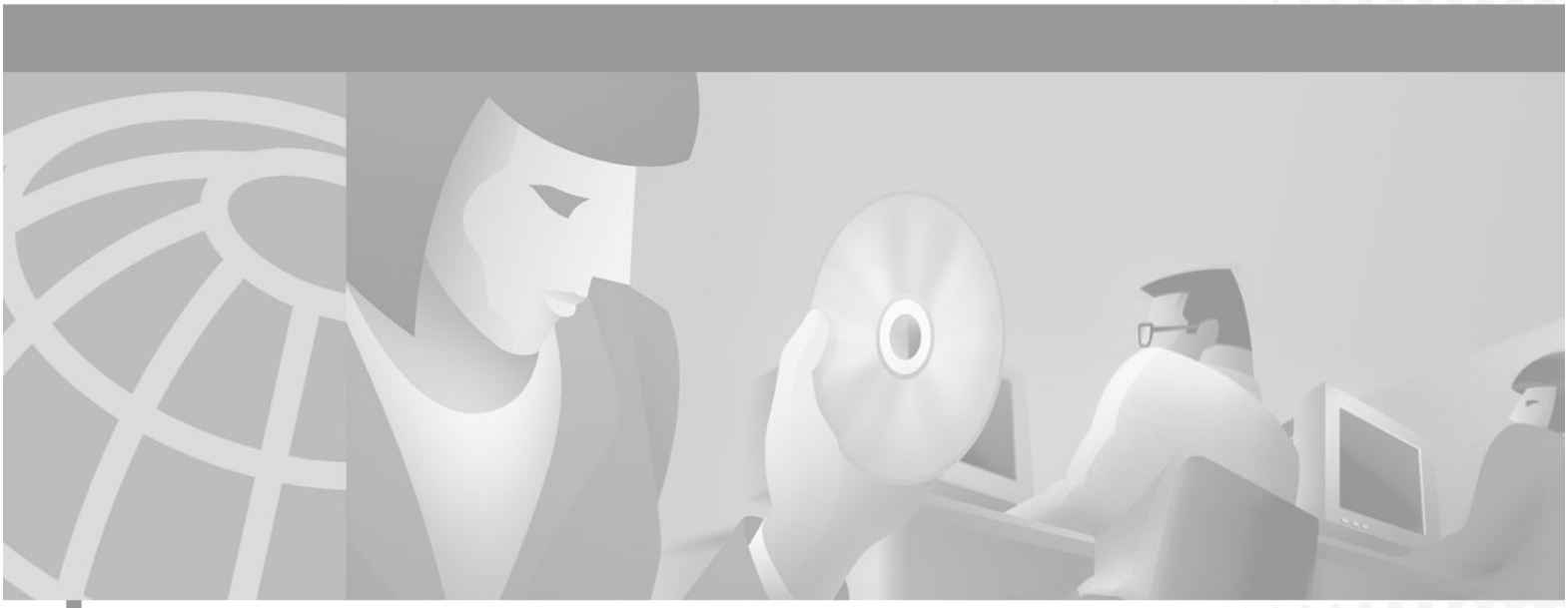


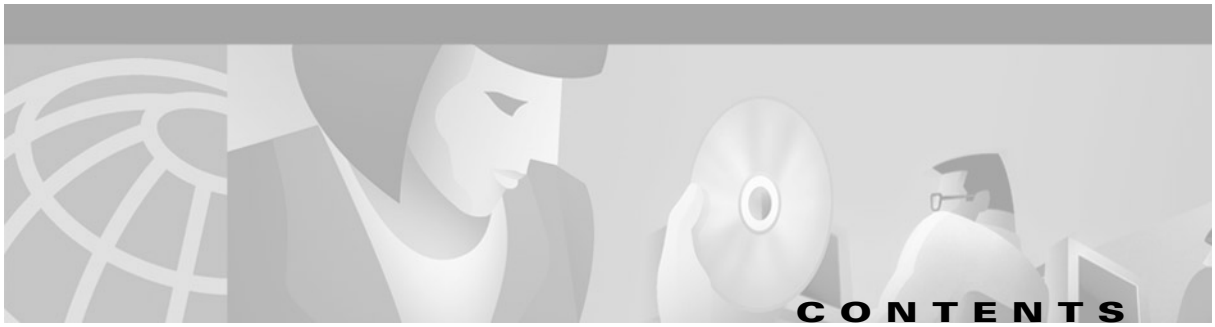
# Cisco Mobile Wireless Fault Mediator 2.1 - Fault Engineering Reference Guide

**Corporate Headquarters**
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 526-4100

Text Part Number: OL-1311-03

*Cisco Mobile Wireless Fault Mediator 2.1 - Fault Engineering Reference Guide*


RiverSoft



# CONTENTS

CHAPTER 1

# Introduction

Welcome to the *Cisco Mobile Wireless Fault Mediator 2.1 - Fault Engineering Reference Guide*. This chapter provides an outline of the other chapters in this guide, which examines the function of Fault Manager. Additionally, it demonstrates the styles used in the guide.

## What this Guide Contains

This document provides a comprehensive overview of *Cisco Mobile Wireless Fault Mediator 2.1 - Fault Engineering Reference Guide*, its function, its core components, and its relationship to MWFM NMOS. The following table provides a summary of the contents of each chapter.

***Table 1-1 Chapter Titles and a Summary of their Contents***

| Chapter | Chapter Title | Description |
|---|---|---|
| 1 | Introduction | Outlines the styles and conventions used in the Cisco documentation set. |
| 2 | Object-oriented Principles and MWFM NMOS | Provides an overview of object-oriented design principles, Active Object Classes (AOCs), the Containment Model, and the MWFM Network Management Operating System (NMOS). |
| 3 | Introduction to Mobile Wireless Fault Mediator | Introduces Cisco's solution for fault management, Mobile Wireless Fault Mediator . It outlines the three main functions of Fault Manager: polling, event management, and presentation of the correlation results to the user. In addition, it describes the extensions Cisco Mobile Wireless Fault Mediator adds to the AOCs for performing the above function. |
| 4 | Polling—The Event Generation Process | Describes the NMOS' polling controller MONITOR in more detail, the process of polling and generating events, and the Polling Agents MONITOR uses for these purposes, together with the first extension Cisco Mobile Wireless Fault Mediator makes to the AOCs—the poll definitions. |

*Table 1-1 Chapter Titles and a Summary of their Contents (continued)*

| Chapter | Chapter Title | Description |
|---|---|---|
| 5 | The Event Management Process | Details the second of the AOC extensions—the event correlation methods—which determine the operation of AMOS, the event correlation engine of Mobile Wireless Fault Mediator. The chapter also describes the four components of the method template (the basis for all event correlation methods). |
| 6 | The Method Conclusion In Action | Looks in more detail at the final component of an event correlation method—the method conclusion. The method conclusion contains actions and method control commands to manipulate databases. |

*Table 1-2 Appendix Titles and a Summary of their Contents*

| Appendix | Appendix Title | Description |
|---|---|---|
| A | AMOS Databases | Lists the column names available in the AMOS Events database, AMOS Entity database, and AMOS Class database. |
| B | The Eval Statement | Partially reprinted from the *Cisco Mobile Wireless Fault Mediator 2.1- Topology and Platform Modeling Reference Guide*, this appendix describes the syntax of the **eval** statement, which enables you to access database records. |

*Table 1-3 Supplementary Matter*

| Chapter Title | Description |
|---|---|
| Glossary | Provides definitions for MWFM-specific terms and details relevant networking and network management terms. |

# Who is this Guide for?

This guide is supplied as support material for people working with Cisco Mobile Wireless Fault Mediator in an administration or technical services capacity. The aim is that the documentation will familiarize you with enough functional information to enable you to configure and manipulate Cisco Mobile Wireless Fault Mediator to suit the requirements of your installation.

It is anticipated that you are already familiar with networking products, concepts and terminology, but may not be conversant with Cisco products or Cisco nomenclature.

# Where Should I Start to Read this Book?

The *Cisco Mobile Wireless Fault Mediator 2.1 - Fault Engineering Reference Guide* commences with an introductory chapter designed to provide the reader with an overview of object-oriented principles, the Containment Model used to analyze the network and the components of MWFM NMOS. If you are familiar with these principles and MWFM NMOS itself, you should commence reading with Chapter 3, "Introduction to Mobile Wireless Fault Mediator," which describes the components that constitute Cisco Mobile Wireless Fault Mediator. Chapters 2 and 4 are more detailed chapters which describe all aspects of network polling, the event correlation methods, and the menu methods. Appendix A, "AMOS Databases," and Appendix B, "The Eval Statement," provide additional reference material in relation to the attributes available in the poll definitions and event correlation methods and the **eval** statement.

# Styles and Conventions Used in this Guide

Several specialized styles used throughout the *Cisco Mobile Wireless Fault Mediator 2.1 - Fault Engineering Reference Guide* highlight important material, key definitions, or terminology. The following sections describe the meaning of these specialized styles.

## Emphasis

Emphasis is used to introduce new terms or important points within the text. It is only used on the first occasion the word or term appears in a section and is denoted by ***bold weighting*** and in italics.

## Command lines

Any command lines appear in the command line type face and are surrounded by a grey text box as displayed below.

```
riv_f_amos -domain <DOMAIN_NAME> &
```

## All caps

All caps is used to easily identify any MWFM NMOS or Fault Mediator component within the text. For instance, DISCO refers to the MWFM NMOS discovery component while AMOS refers to the Cisco Mobile Wireless Fault Mediator event correlation engine.

## OQL keywords

Keywords of OQL, the SQL-like language provided by MWFM, are highlighted in bold text, such as the example of **eval** above.

# Conclusion

This chapter has introduced you to the *Cisco Mobile Wireless Fault Mediator 2.1 - Fault Engineering Reference Guide*, suggested where you should commence your reading, and indicated how to interpret the text conventions used herein.

The next chapter introduces object-oriented principles and the other underlying principles behind MWFM NMOS—the heart of Cisco's solution to network management issues.

CHAPTER 2

# Object-oriented Principles and MWFM NMOS

This chapter provides an introduction to object-oriented principles and the Active Object Classes, the basis for the rest of this guide. This chapter also provides an overview of the MWFM Network Management Operating System (NMOS) which is the foundation for the Cisco product suite and fundamental to the operation of Cisco Mobile Wireless Fault Mediator .

## MWFM Fundamentals

This chapter is designed as an introduction to object-oriented principles which are fundamental to understanding the operation of MWFM. It also introduces the Active Object Classes (AOCs) and the Containment Model, which MWFM NMOS uses to manage network devices, and provides an overview of the architecture of MWFM NMOS and how its components interact with each other to provide an underlying layer for Cisco Mobile Wireless Fault Mediator.

## Principles of Object-oriented Programming

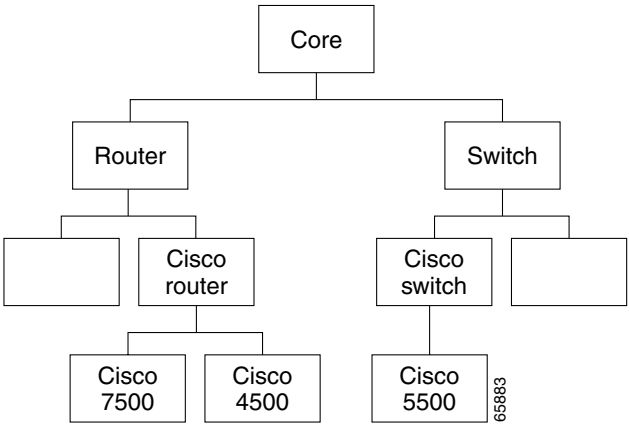
Cisco Mobile Wireless Fault Mediator follows the conceptual principles of object-oriented programming, which is organized around *objects* rather than a series of rules or actions. In network management terms, objects are devices or entities (pieces of hardware) on the network. These include routers, switches, hubs, chassis, interfaces, cards, ports, etc.

**Note** The terms object and device are used loosely in this guide. This is because some objects represent actual devices while others represent interfaces and connections. There are also abstract objects such as containers, which are introduced shortly.

These objects are grouped into classes, organized into a hierarchical structure, and managed on the basis of their location within the structure. A class of objects is simply a group of objects that can be managed the same way because of common attributes.Grouping objects into classes introduces a hierarchical structure, which ranks classes into levels of subordination based on how many attributes are shared on each level. This type of structure also invokes a principle called inheritance.

Inheritance means that any object classes located lower in the hierarchical structure assume the characteristics (or management policies) of those object classes superior to it. The accepted means of referring to object classes is as a parent-child relationship; the child class (or sub-class) inherits characteristics from its parent, the parent class (or super-class). Any management policies defined in a parent class are adopted by the child class without having to specify them again. Figure 2-1 demonstrates the concept of a hierarchical structure.

***Figure 2-1 Arranging network entities into a hierarchical structure***




The illustration shows that all classes inherit from a single root, the class called Core. Consider a Cisco 7500 router class; this type of router has unique characteristics that distinguish it from a Cisco 4500 class router. For instance, it may have more sensitive environmental characteristics that need to be managed differently; thus, it is necessary to have a series of management policies dedicated to the Cisco 7500 router class. However, both the Cisco 4500 and 7500 are manufactured by Cisco and would therefore share vendor-specific MIB information. This common information could be specified in the Cisco Router class. Additionally, both are routers and therefore part of the Router class, which may contain management policies dedicated to handling general routing table information. Finally, all routers are in turn child classes of the Core class. Management policies in the Core class are inherited by all network entities and typically include generic policies. For example, to ensure the integrity of the network it may be necessary to check the availability of all network devices every 60 seconds via a ping poll. This type of action could be defined in the Core class and inherited by all other devices.

Finally each object in a class is referred to as an *instance* of that class. Devices such as routers, switches, cards, etc. are objects and thus, all instances of object classes. In the example shown in Figure 2-1, any Cisco 7500 router is an instance of the object class “Cisco 7500”.

## Summary of Object-oriented Concepts

The following list provides a summary of the most important object-oriented concepts to grasp.

- All devices/entities are objects.
- Classes are groupings of objects based on predefined criteria.
- Classes are arranged into a *hierarchical* tree structure which has a single root. The principles of *inheritance* apply to this structure; thus, descendents of a class inherit the management policies associated with the parent class.
- Every object is an instance of a class.

An extension to the concepts of objects and classes is that of Active Object Classes, which is the next topic for discussion.

# Active Object Classes

A key feature of MWFM NMOS and Cisco Mobile Wireless Fault Mediator is the use of Active Object Classes (AOCs). AOCs are based on the concepts of object classes and object modelling, but the object classes in the hierarchy are active, which means that upon instantiation of an object to a class the properties, behavior and management policies contained in the relevant AOC are automatically applied to the object without configuration by the user.

**Note** Instantiation is the act of creating a new instance of an object class and applying the properties, behavior, and management policies associated with that class of object.

## What do the AOCs Contain?

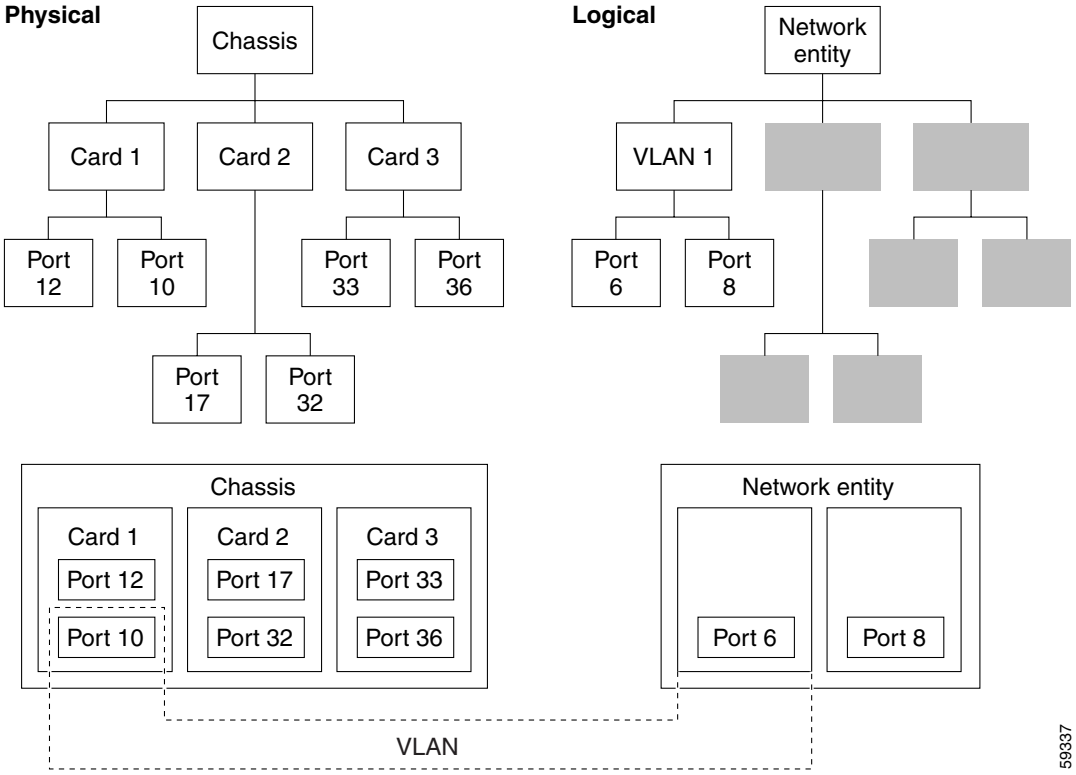
The AOCs are fundamental to the operation of MWFM NMOS and Cisco Mobile Wireless Fault Mediator, so it is important to have an understanding of their content. This section lists the contents of the AOCs and introduces some of the components and terminology which are expanded upon in more detail later in this guide. An AOC contains the following:

- The name of the object class.
- The parent class (or super-class) of the AOC.
- The rules to instantiate a device to an AOC, i.e., how the system determines the class to which an object belongs.
- A reference to the appropriate Data dictionary(s). The data dictionary is like a library. It contains all of the ASN.1 descriptions of all of the datatypes used by the Active Object Class.
- The extensions, which are network management application-specific policies. Cisco Mobile Wireless Fault Mediator has a series of extensions associated with it that include:
  - The poll definitions (see Chapter 4, “Polling—The Event Generation Process”)
  - The event correlation methods (see Chapter 5, “The Event Management Process”)
  - The menu methods
- The visual menus for MWFM NMOS.

# The Containment Model

In networking terms, containers are objects that can "hold" a collection of other objects or entities, some of which may themselves be containers. The MWFM NMOS Containment Model reflects the real world topology of the network that is being modelled in both a physical and a logical sense. Figure 2-2 shows two ways of applying this containment principle, a physical hierarchy and a logical hierarchy.

***Figure 2-2 The Containment Principle using Network Objects***




The physical hierarchical Containment Model shows that a switch chassis can "contain" network interface cards, power supplies, and software relating to the internal operating system. The cards in turn can "contain" ports.

The logical hierarchical Containment Model shows that a network also contains VLANs, which contain ports. Logical containment is particularly useful in root-cause analysis. For example, if Port 10 on Card 1 is associated with a central server, the failure of this server will affect a user connected to Port 6. Without the ability to incorporate this logical aspect it would be time-consuming to determine that the failure of the central server was also the cause of the problems experienced by the user connected to Port 6 (particularly if they are in different geographical locations).

# The Architecture of MWFM NMOS

The concepts described in the first part of this chapter form the basis of MWFM NMOS, which in turn forms the foundation of the MWFM network management suite. With MWFM NMOS installed, other applications or tools can be integrated to form a complete network management solution.

MWFM NMOS is made of eight components which interact to enable all devices on a network to be discovered and an accurate network topology model developed. Table 2-1 provides an introduction to each MWFM NMOS component, as well as a description of its function.

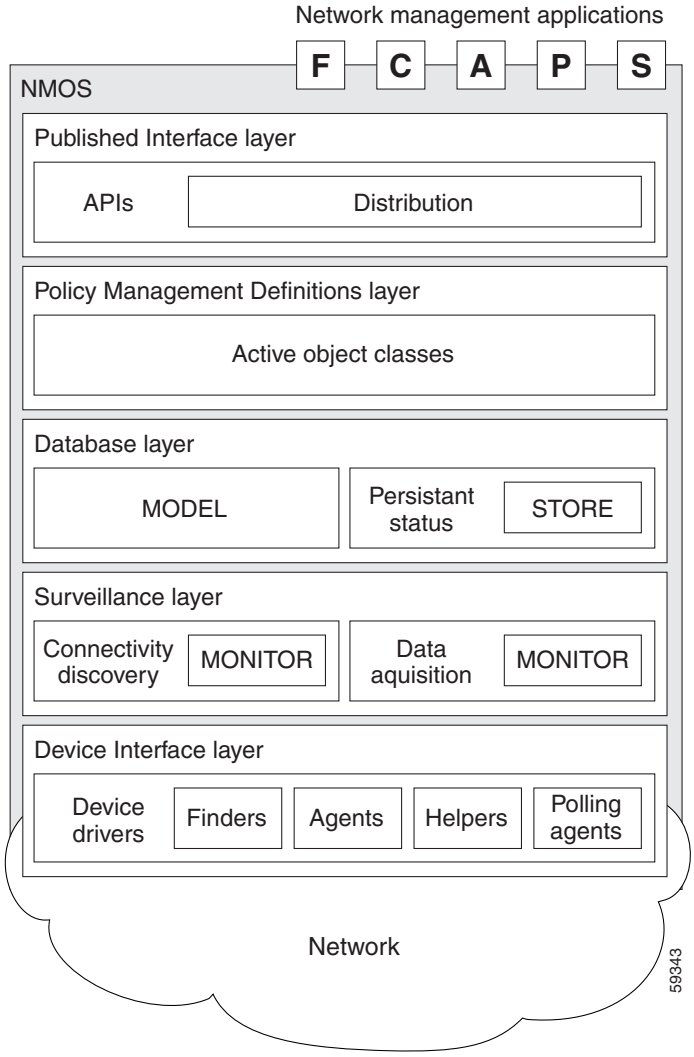
***Table 2-1 Synopsis of MWFM NMOS components***

| MWFM NMOS component and executable name | Function |
|---|---|
| AUTH (riv_auth) | The authentication engine. AUTH monitors user security privileges and ensures no unauthorized transactions take place within MWFM NMOS. |
| CLASS (riv_class) | The Active Object Class management and distribution system. CLASS is a dynamic library that manages the AOCs, which contain the criteria for instantiation of network devices and any management policies associated with these devices. |
| CTRL (riv_ctrl) | The process controller. CTRL initiates and monitors the status of all MWFM NMOS components. |
| DISCO (riv_disco) | The auto-discovery process controller. DISCO discovers device existence and connectivity via a series of Finders, Discovery Agents and Helpers. The retrieved information is knitted together to form an accurate network topology model by processes called Stitchers. |
| DIST (riv_dist) | The distribution engine. DIST enables MWFM NMOS to communicate with third party applications and the distribution of MWFM NMOS into multiple domains. |
| MODEL (riv_model) | The network topology distributor. MODEL is both the repository for the discovered network topology, and the distributor for any application that requires it. |
| MONITOR (riv_monitor) | The polling process controller. MONITOR is responsible for monitoring the status of devices within the discovered network topology. MONITOR uses a series of Polling Agents—whose polling methodology information is held in the AOCs—to poll the network periodically to determine whether there have been any state changes; as a result of discovering such changes, an event stream is generated which can be sent to other processes for correlation. |
| STORE (riv_store) | The persistent storage engine and historical archive. STORE qualifies events as topology updates or those generated by the Polling Agents and stores them in a central repository for distribution to other components if required. |
| OQL (riv_oql) | The Object Query Language service provider and command line interpreter. OQL can communicate with and interrogate the databases of MWFM NMOS. |

# How MWFM NMOS Components Interact

The components of the MWFM NMOS described in Table 2-1 each perform specific functions to provide a complete network management solution, as in Figure 2-3.

***Figure 2-3 The Components of MWFM NMOS***




In order to provide a full network management system, the components must share information. At the end of the discovery process conducted by DISCO using Finders, Discovery Agents, Helpers and Stitchers (to resolve device connectivity), a completed network topology is created and transferred to MODEL. MODEL then consults the AOCs through CLASS and each device on the network is instantiated to a particular object class based on the AOCs' criteria. MODEL stores the final topology in a database for distribution to other MWFM NMOS components.

MONITOR can then poll the network for state and information changes via a series of Polling Agents using the network topology from MODEL and the polling strategy for each device class from the AOCs, via CLASS. MONITOR sends the polling strategy to the Polling Agents, who then poll the network for information. The information retrieved by the Polling Agents is sent to MONITOR, which uses it to

generate a series of events (depending on how the poll definitions have been configured) and distribute this event stream to any network management applications running on top of the NMOS platform (for instance, Cisco Mobile Wireless Fault Mediator).

STORE can be configured to listen to all information such as events and alerts generated by the interaction of other NMOS components. These events can be created by state changes of devices, determined by MONITOR, or by state changes in topology, through information received from MODEL. This information is placed in a series of databases so that it is available in other NMOS sessions.

To enable MWFM NMOS components to communicate with third-party applications such as help desk or trouble ticketing systems, or even other MWFM NMOS domains, DIST can use a series of configurable listeners to filter information to the particular application.

**Note** A MWFM NMOS domain is not an internet domain. It is a name assigned to a particular view of the network. When MWFM NMOS starts, a domain is always specified.

CTRL is the master process run before all other processes; it is used to manage the function and operation of all other MWFM NMOS components.

Policing the communications between all of these processes is AUTH, which supervises all transactions to ensure that the current user has the required access and user privileges to manage and manipulate the network.

# Conclusion

This chapter has explored the fundamentals of object-oriented programming principles, the AOCs, the Containment Model, and MWFM NMOS. You have also seen the architecture of the NMOS and how its components share information amongst themselves, other Cisco applications, or even third-party applications. Cisco Mobile Wireless Fault Mediator makes extensive use of the NMOS' functions for fulfilling its tasks, and how it does so is explored in Chapter 3, "Introduction to Mobile Wireless Fault Mediator."

CHAPTER 3

# Introduction to Mobile Wireless Fault Mediator

This chapter introduces Mobile Wireless Fault Mediator itself in detail. It sets the scene for subsequent chapters by establishing the components of Mobile Wireless Fault Mediator, how they relate to each other, and how they operate.

## Mobile Wireless Fault Mediator

Mobile Wireless Fault Mediator is a solution for the fault management area identified in the ISO FCAPS network management model. Mobile Wireless Fault Mediator is designed as an application that runs on top of MWFM NMOS and adds fault management capabilities to the MWFM product. Mobile Wireless Fault Mediator uses many of the core processes in MWFM NMOS to facilitate its primary functions, fault detection, and root-cause analysis.

**Note** Root-cause Analysis is the process of pinpointing the exact reason and location of alerts on the network. This is one of the fundamental objectives of Mobile Wireless Fault Mediator.

Mobile Wireless Fault Mediator is essentially an event processing and correlation engine; it uses MWFM's network topology model of the network to correlate a stream of network events (generated as a result of polling) according to the location of the device in the network and the class of device on which the event occurred. As a result of this correlation process, Mobile Wireless Fault Mediator can upgrade the most severe events to alerts.

**Note** An event is a message indicating there has been an occurrence on the network, e.g., a device is unreachable. An alert is a correlated event, i.e., an even that has been deemed significant enough to warrant the attention of the network administrator.

# The Functions of Mobile Wireless Fault Mediator

Mobile Wireless Fault Mediator's operations can be classified into three functional areas:

1. Polling—the event generation process, controlled by the poll definitions.
2. Event management—controlled by the event correlation methods.
3. Object interaction—controlled by the menu methods.

The poll definitions, event correlation methods and menu methods for each instantiated object reside in the AOC extensions, which are installed at the same time as Mobile Wireless Fault Mediator.

# Mobile Wireless Fault Mediator's AOC extensions

Each of the AOC extensions have distinct functions and responsibilities as part of Mobile Wireless Fault Mediator's operation.

## Polling and the Poll Definitions

The manner in which network polling is undertaken is determined by the first of the AOC extensions, the poll definitions. The poll definitions have two functions:

1. They determine how MONITOR and the Polling Agents will poll a particular object class in the network. For instance, how often an object is polled, the type of Polling Agent employed to do the polling, the frequency of the polling, what information is sought during the polling process, what traps to listen for, etc., are all specified in the poll definitions.
2. They determine the nature of the raw events generated (and the attributes associated with these events) when a device undergoes a state or performance change. These raw events make up the event stream, which is the input to the Active Managed Object Store (AMOS), the event correlation engine.

## Event Management and the Event Correlation Methods

The next set of AOC extensions is associated with the management and interpretation of the event stream by AMOS and is called the event correlation methods.

**Note** The event stream is a series of raw events propagated by MONITOR as a result of the poll definitions and the operation of the Polling Agents.

The event correlation methods define how AMOS responds to the events generated for each object class as a result of the polling strategy assigned to that class. For example, if a ping poll to a particular instance of an object class fails, an event is generated. The event correlation methods define how AMOS should interpret the event generated as a result of the failed poll attempt, according to that object's class.

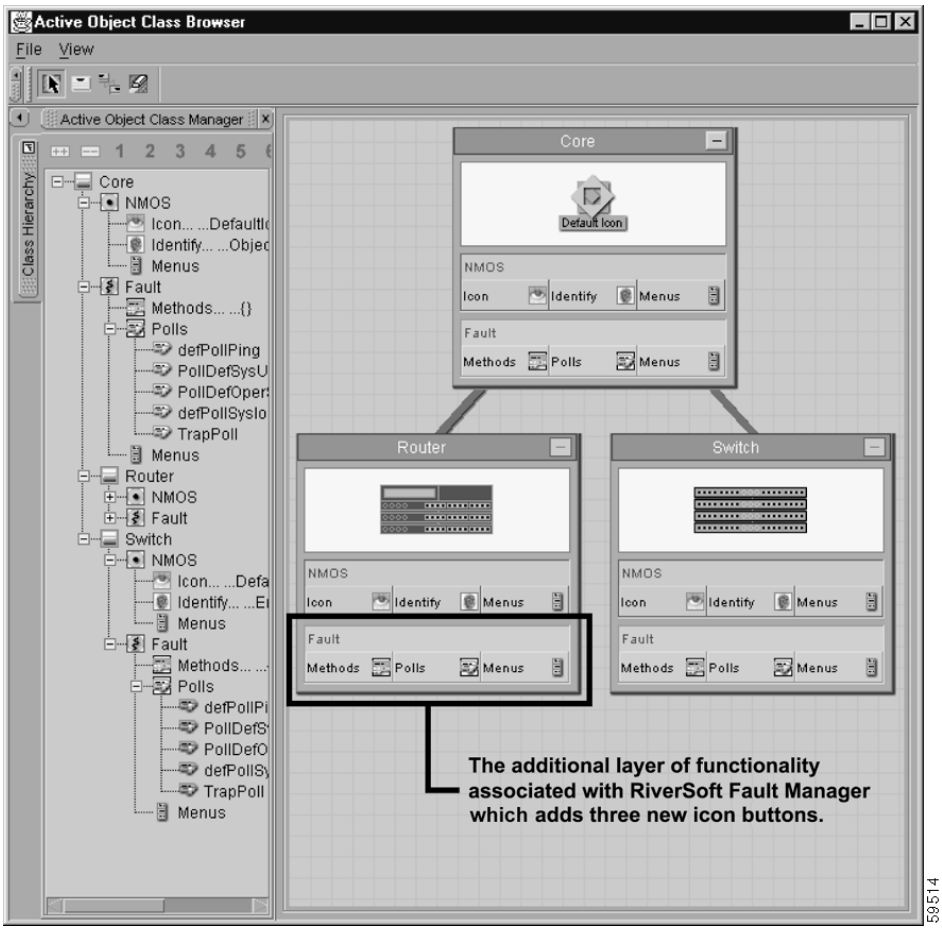
# Accessing the AOC extensions—the AOC browser

The Active Object Class (AOC) browser is a standalone GUI application using Java that allows the AOCs to be accessed, edited and manipulated. The fundamentals central to the function of Mobile Wireless Fault Mediator are explained in this section.

When the AOC browser is launched, the browser's Main View appears. There are two areas in the Main View, the Main Work Area and the Class Hierarchy Tree. The Main Work Area displays a graphical representation of the AOC hierarchy. The Class Hierarchy Tree appears to the left of the Main Work Area and displays the AOCs in a tree structure. With MWFM NMOS running, each class in the Main Work Area has three dialog buttons associated with it. These are:

- The Icon button—opens the Icon Open dialog, which enables the appearance of the class in the GUI to be edited.
- The Identify button—opens the Filter Builder dialog, which enables the instantiation rules in the AOC to be constructed.
- The Menus button—opens the Menu Builder dialog, which allows the menu actions appearing in the GUI to be constructed.

***Figure 3-1 The Main View of the AOC browser as it appears with just MWFM NMOS installed***

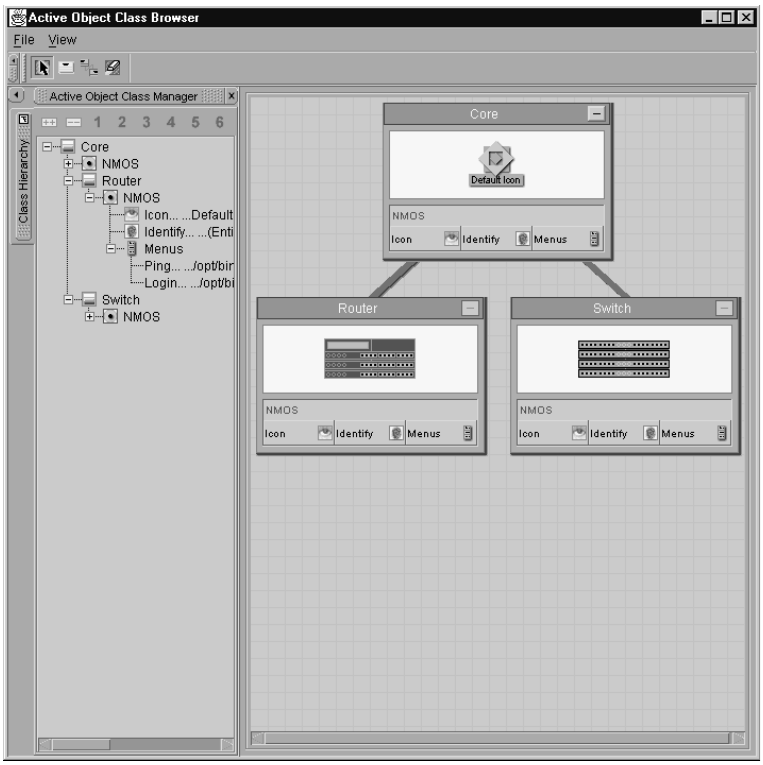


When Mobile Wireless Fault Mediator is installed it introduces three new icon buttons to the AOC browser, associated with its AOC extensions and the additional layer of functionality they introduce. The three additional buttons are:

- The Polls button—opens the Poll Editor dialog, which enables the poll definitions to be constructed.
- The Methods button—opens the Event Method dialog, which enables the event correlation methods to be constructed.
- The Menus button—operates in the same way as the Menus button associated with MWFM NMOS, except it opens a Menu Builder dialog unique to Mobile Wireless Fault Mediator. This allows the menu methods, which affect the menu actions appearing in the GUI, to be configured.

An example of the AOC browser with the additional Mobile Wireless Fault Mediator buttons can be seen in Figure 3-2.

***Figure 3-2 The AOC browser as it appears with Mobile Wireless Fault Mediator installed***



# Mobile Wireless Fault Mediator Event Correlation Engine

The heart of Mobile Wireless Fault Mediator is AMOS, the event correlation engine. AMOS has access to the event stream generated by MONITOR (via the Polling Agents and the poll definitions in the AOCs) and information about the topology of the network, and relates the two of these together. It also knows about the behavior of classes of devices and it knows to which class a given instance in the network topology belongs.

AMOS therefore interprets the event stream associated with a given instance of a device in the network topology according to the event correlation methods that govern the behavior of Mobile Wireless Fault Mediator for devices of that class. It can also interpret the effect that a given event on a device will have on the devices connected to it, using the connectivity information from the network topology.

For instance, a series of “pingFail” events on a particular device may mean the user should be warned about a problem associated with the device. AMOS, using the event correlation methods for that device, creates an alert. From the topology information and Containment Model details, AMOS is also able to determine that five other devices will be affected by the original failure, alert the user of this, and suppress (ignore) any further events from these dependent devices until the “pingFail” error on the original device is rectified.

To facilitate this process, AMOS uses a series of databases to store information from MWFM NMOS components—cached information enabling AMOS to function efficiently. AMOS holds a record of the correlated event stream in the AMOS Events database and a version of the network topology in the AMOS Entity database. AMOS also reads information pertaining to the AOCs from CLASS and stores it in the AMOS Class database. These databases and the fields they contain are fully described in Appendix A, “AMOS Databases.”

# Interaction Between Mobile Wireless Fault Mediator and MWFM NMOS

Mobile Wireless Fault Mediator does not operate in isolation; it relies on interaction with many of the MWFM NMOS components (CLASS, MODEL, MONITOR, CTRL, STORE and DIST). This section provides an overview of the interaction between Mobile Wireless Fault Mediator and other MWFM NMOS components. Also, it includes a list of the prerequisites to be completed before AMOS can be started, as well as information on how to start AMOS.

# Prerequisites for running AMOS

- Task one—ensure that DISCO, the discovery process controller residing in MWFM NMOS, has been run and has successfully completed a network discovery. The network topology and containment model must have been passed to MODEL, the network topology distributor.
- Task two—ensure that MODEL is running, since AMOS loads topology information from MODEL into the AMOS Entity database.
- Task three—ensure that CLASS, the AOC management system, is running, since MONITOR loads up the poll definitions created in the AOCs (via the AOC browser), which reside in CLASS.
- Task four—ensure that both MONITOR, the polling process controller is running and the Polling Agents have been launched from the command line. It is the Polling Agents of MONITOR that are responsible for generating the event stream.
- Task five—ensure that STORE, the persistent storage engine, is running, as it is STORE that listens to the event traffic and acts as the historical repository for topology changes and event updates. When AMOS starts, it loads information from STORE into the AMOS Events database.

## Configuring the AOC Extensions

Any customizing of the AOC extensions associated with Mobile Wireless Fault Mediator is best done prior to launching AMOS, by launching the appropriate editor from the AOC browser—the Poll Editor dialog for the poll definitions, the Event Method dialog for the event correlation methods, or the Menu Builder for the menu methods. This step is optional, since the system will function with the default AOC extensions if they are not specifically customized.

# Starting AMOS

AMOS is started using the following command line:

```
riv_f_amos -domain <DOMAIN_NAME> &
```

| | |
|---|---|
| riv_f_amos | The executable name for the Mobile Wireless Fault Mediator event correlation engine, AMOS. Other Cisco network management applications may have event correlation engines, so it is necessary to distinguish between them by adding an additional letter unique to the application executable name, in this case “f” for Mobile Wireless Fault Mediator. |
| <DOMAIN_NAME> | Corresponds to the name of the domain under which the MWFM NMOS processes are being run. |

# Process Flow When AMOS is Launched

This section expands upon the previous description of the interaction between Mobile Wireless Fault Mediator and NMOS. It also presents an overview of the information flow when AMOS is launched. This should enable some of the concepts introduced in this chapter and earlier chapters to be consolidated, prior to embarking on a more detailed analysis of each of the AOC extensions.

When AMOS is started, the following sequence of events takes place:

1. Information is loaded into the AMOS databases from other NMOS components.
   a. Event information is loaded from the databases of STORE into the AMOS Events database.
   b. Network topology and Containment Model information are loaded from the databases of MODEL into the AMOS Entity database.
   c. Class-based information, including the AOC extensions (the poll definitions, event correlation methods and menu methods) associated with Mobile Wireless Fault Mediator, is loaded from CLASS into the AMOS Class database.
2. AMOS listens for events generated as a result of the polling process undertaken by MONITOR.
3. Any raw event generated as a result of the polling process is associated with a device, which is an instance of a class. When a raw event is received by AMOS it uses the event correlation methods associated with that class (from the AMOS Class database) to correlate the received event with the network topology and Containment Model residing in the AMOS Entity database.
4. Throughout the correlation process, STORE listens for newly generated events and updates. AMOS then injects these new events or amended event records into the databases of STORE to ensure its historical record of network events is complete and up-to-date.

# Conclusion

This chapter introduced Mobile Wireless Fault Mediator, which manages three main functions: polling, event management and object interaction. Each of these areas has an AOC extension associated with it which can be accessed via the AOC browser in the GUI. The AOC extensions are: the poll definitions, which define the event input stream to AMOS; the event correlation methods, which define how AMOS should interpret and process the event stream; and the menu methods, which enable user interaction with an instance of an object.

In addition, the prerequisites for running AMOS, the command line to start AMOS, and the process flow when it is launched have all been introduced.

The next chapter is the first of the more detailed chapters that explore the attributes of the AOC extensions associated with Mobile Wireless Fault Mediator. It provides an in-depth analysis of the event generation process caused by the interaction between the Polling Agents of MONITOR and the network.

CHAPTER 4

# Polling—The Event Generation Process

This chapter describes polling—the event generation process. It investigates the polling process controller MONITOR, and the Polling Agents MONITOR uses to monitor the status of network devices. In addition, it details the first of the AOC extensions; the poll definitions, which determine how the Polling Agents poll network devices and the contents of the events generated as a result of the polling process.

## Monitoring the Network

The NMOS polling process controller MONITOR controls network monitoring. MONITOR uses a series of Polling Agents that interact with the network, generating events depending on poll definitions in the AOCs. These events are the inputs to Cisco Mobile Wireless Fault Mediator and are correlated with topology information from the NMOS component MODEL by the event correlation engine, AMOS. AMOS can then promote these events to alerts, depending on the event correlation methods employed.

Even though MONITOR is part of MWFM NMOS, it has a specific role within Cisco Mobile Wireless Fault Mediator, namely, to provide an event stream to AMOS. To enable MONITOR to perform this role, Cisco Mobile Wireless Fault Mediator provides MONITOR with poll definitions to control the Polling Agents.

MONITOR receives this information via CLASS (which maintains the AOCs) and distributes the poll definitions and polling methodology to the appropriate Polling Agent. The Polling Agent uses this information to form the basis of its polling strategy. The network is polled and the results are then returned to MONITOR, which in turn broadcasts the information back to the network management application that requested it—in this case, Cisco Mobile Wireless Fault Mediator.

## Polling Agent Classification

The Polling Agents are responsible for retrieving requested information from the network. There are a number of Polling Agents available, each one specialized to use one of the network protocols used for data retrieval, though they fall into two main categories, Synchronous and Asynchronous. Once configured, each Agent stores a cache of its polling strategy so that it does not need to continually query MONITOR for operational information, making the system more efficient.

## Synchronous Polling Agents—timer-based

There are two synchronous Polling Agents—the Ping Polling Agent and the SNMP Polling Agent—which poll the network according to a Frequency attribute that is part of the poll definition; for example, pinging a device every 30 seconds.

### Ping Agent—polling for existence

The ping process is the most common way to check that a device is available from another location in the network. The Ping Polling Agent checks that a device is still present, live and contactable in the network (using ICMP), by sending a packet of information on a periodic basis to an IP address and waiting for a response. On many network devices pings are typically run as a low priority and often timeout; for this reason, Ping Polling Agents are generally configured to send several pings to a device before generating an event announcing loss of connectivity.

### SNMP Agent—polling for details

The SNMP Polling Agent is used to collect information—defined in the MIB—from devices on the network that use the SNMP protocol.

## Asynchronous Polling Agents—interrupt based

By default there are two asynchronous Polling Agents—the Trap Agent and the Syslog Agent. Asynchronous means that the Agents are constantly on standby, monitor the network continuously, and are activated by the receipt of relevant SNMP traps or Syslog messages they are configured to listen for.

### Trap Agent—listening for SNMP traps

The Trap Polling Agent is an asynchronous Agent that listens for traps. The types of traps it listens for and the responses to these traps are defined in the poll definitions.

### Syslog Agent—polling for messages

The Syslog Polling Agent monitors the system log (Syslog) files listening for updates. If an update occurs, the Syslog Polling Agent searches through the system file for details it is interested in to determine whether changes have occurred.

# Outputs from the Polling Process—An Event Overview

To enable successful and efficient event processing (the subject of the next chapter) it is important to have a fundamental understanding of the output from the polling process, i.e., the types of events produced as a result of polling (which by nature is an interrogatory or query process) and practical situations that could generate these events.

Events from the polling process can be classified into two categories: those based on state changes of network entities and those which are information-based.

## State Change Events

State change events are initiated when something in the network changes state; for example, a network device loses power, a WAN link goes down making a series of devices unreachable, etc. These types of events are generally associated with Ping Polling Agents who poll devices for availability rather than extracting information. For this reason, another term that is used for this category of polling is availability polling. In a sense, trap events could also be considered in this category since a linkUp trap indicates an obvious state change.

## Information-based Events

Information-based events are those associated with utilization and include any factor that can indicate network degradation—for instance, sysUpTime, the number of errors on a particular link, bandwidth usage exceeding defined thresholds, etc. These aspects typically involve SNMP and Syslog Polling Agents, although Syslog Polling Agents are also associated with state change events.

## A Final Word on Events

There is a final key point that must be mentioned in regard to poll definitions and events. “Events” can also be generated by network devices themselves, in the case of SNMP notifications and Syslog messages. However, these “events” will not be recognized by Cisco Mobile Wireless Fault Mediator unless a poll definition is constructed to interpret them. This is a vital point to take from this chapter; the fact that polling (based on the accurate configuration of the poll definitions) is the main way that an event can be generated, interpreted and broadcast to AMOS for correlation.

There are two alternative ways to generate an event. Firstly, the MWFM NMOS component DIST can construct events and broadcast these to other components such asCisco Mobile Wireless Fault Mediator. Secondly, any application that can implement OQL statements can be used to construct an event which can be inserted into the AMOS Events database. However, polling remains the primary means of event generation.

# Poll Outcome

Before launching into a detailed analysis of the poll definitions and how they operate, it is appropriate to consider what the outcomes of the polling process are. This places some perspective on the remainder of the chapter, since it is these outcomes (or events) which are broadcast to AMOS in the event stream.

Polling results in four possible outcomes:

1. Poll success—device contactable, threshold condition evaluates false, no event generated.
2. Threshold condition—the threshold condition specified in the poll definition evaluates true.
3. Poll fail condition—device unreachable, or the Polling Agent failed to retrieve the prescribed MIB variables or Syslog information specified in the threshold condition within the timeout period.
4. Poll restore condition—device contactable, after previously being unreachable or after a threshold condition which was previously true evaluates false.

However, only three of these outcomes can generate events: the threshold condition, the poll fail condition and the poll restore condition. Exactly which type of outcome, and hence the type of event generated, depends on the construction of the poll definition. In addition, only one of these outcomes can exist at any one time. It is not possible to generate an event specifying that both a threshold and restore condition exist as they are independent outcomes.

# Poll definitions

Crucial to managing the network is the process of information-gathering from devices on the network. As mentioned in Chapter 3, "Introduction to Mobile Wireless Fault Mediator," the poll definitions have two important functions.

Firstly, they define how, when and where MONITOR and the Polling Agents poll the network. For instance, how often an object is polled, the type of Polling Agent employed to do the polling, the frequency of the polling, what information is sought during the polling process, what traps to listen for, etc.

Secondly, they define the construction of the events (event records) generated as a result of the polling process due to changes in network devices. These events constitute the event stream and represent the input to AMOS; they may subsequently be upgraded to alerts depending on the event correlation methods.

The poll definitions reside in the extensions of the AOCs added by Cisco Mobile Wireless Fault Mediator and are constructed or customized using the Poll Editor in the AOC browser.

Like other attributes of the AOCs and the class-based system, the poll definitions follow a hierarchical structure; therefore, any poll definition defined in a parent class is applied to a child class unless locally redefined.

## The poll list

The poll list is a series of attributes available for constructing poll definitions. The attributes available in the poll list reflect the function of the poll definitions: to define the polling strategy and content of events in the event stream. There are three distinct areas of attributes in the poll list. They are:

1. The attributes associated with the polling strategy.
2. The attributes associated with the construction of events.
3. The attributes associated with overriding existing event column names.

### Poll strategy list

The poll strategy list is a list of attributes associated with the polling strategy. These are listed in Table 4-1. As identified above, the poll strategy list determines how, when and where the Polling Agents will interrogate the network for information pertaining to a device's state (availability) or aspects associated with its performance.

*Table 4-1 The Attributes Available for Constructing the Polling Strategy*

| Attribute | Description |
|---|---|
| PollDefState | Not currently used, assigned a value of 0. |
| PollName | Defines the name of the poll definition in the AOC. Any string of text is a valid entry; however, it must be unique within the specified AOC. For example, `defPollPing` is a valid PollName. |
| AgentName | Defines which Agent will send and receive the poll: Ping, SNMP, Trap or Syslog (or any other user-defined Agents). |
| Frequency | Determines the rate at which polling should occur. This is defined by a time in days, hours, minutes, and seconds. The frequency attribute is only valid for the Synchronous Agents, Ping and SNMP. |
| Ttl | An abbreviation for time-to-live; it determines how many polls the Polling Agent conducts while in an active state. Only positive integer values are valid. Not currently used. |
| Timeout | Specifies the time to wait for a response from a device. If a device does not respond during the timeout period there are two possibilities:<br>• Generate an event—specifying an appropriate name in the EventName column name, for instance “pingFail”<br>• Re-poll the device based on the details specified in the “AgentExtensions” attribute, for instance, the condition may be poll a device “n” times, then generate an event if it does not respond.<br>The timeout only has relevance to the Synchronous Agents; Ping and SNMP. |
| Threshold | The Threshold condition is used in two ways. Firstly, it is used to specify the parameters to poll for in the case of SNMP, Trap and Syslog Agents. Secondly, it defines a logical condition which determines whether an event is generated or not. If the threshold condition succeeds, i.e., evaluates ***true***, an event is generated and broadcast to AMOS. If it does not succeed, i.e., evaluates ***false***, no event is generated. “SysUpTime > 12,000” is an example of a valid Threshold condition (where 12,000 represents time ticks in hundredths of a second, i.e., 120 seconds). In this case the Agent polls for the SysUpTime and then evaluates the retrieved value against the Threshold test of greater than 12,000. |
| Scope | Allows you to further restrict the execution of the poll to specific instances of Active Objects (or entities) by using column names associated with the entity. The actual information available within the scope condition is any column name contained within the MODEL `master.topoCache.entityByName` database (column names of this database are available in the Cisco Mobile Wireless Fault Mediator 2.0 - topology and Platform Modeling). |
| AgentExtensions | Attribute currently not used. |
| Subject | Allows you to specify which subjects are able to receive the events generated as a result of the polling strategy employed by the Polling Agents. |

## Event construction list—the base generated event

The attributes associated with the construction of an event are listed in Table 4-2. The event constructed using these attributes is known as the base generated event. This information will be present in the event record that is sent to AMOS in the event stream unless it is overridden by one of the event override lists.

*Table 4-2 The Attributes Available for Constructing Events*

| Attribute | Description |
|---|---|
| Severity | Specifies the event severity status of the event record sent to AMOS. In ascending order of severity (except for the final value, None), the possible values are:<br>• Clear<br>• Unknown<br>• Warning<br>• Minor<br>• Major<br>• Critical<br>• None |
| Description | Allows a textual description to be assigned to the event. |
| Location | Identifies the location of the associated device in the network when the event is generated; for example, “Main Server”. |
| Contact | Identifies the person to contact when the event is generated; for example, “Call network administrator”. |
| EventName | Defines the name of the event being generated; for example, “pingFail”. |
| ClassName | Identifies the name associated with the Active Object. |

## Event override lists

As identified earlier in the "Poll Outcome" section on page 4-3, three types of event are generated as a result of the polling process. Each event type has an override list associated with it; these are listed below with a complete description given in Table 4-3 on page 4-8.

1. A threshold condition event—Thresholdlist.
2. A poll fail condition event—Pollfaillist.
3. A poll restore condition event—Restorelist.

The override lists are activated when a threshold condition, poll fail condition or poll restore condition occurs. These lists allow the network administrator to "override" the column names in the base generated event. Essentially, the resulting event record will be a union between the information contained in the base generated event and the information in the relevant override list. If a common column name is specified then the column name in the override list takes precedence.

As an example of the mechanism, consider a situation where the "Description" column name within the base generated event is defined as:

```
Description = ["sysUpTime has been exceeded"]
```

If a poll fail condition occurs then the override capability of the pollfaillist can be employed. If the Description column name is redefined in the pollfaillist as:

```
Pollfaillist = ["Description = 'Poll Failure'"]
```

Then "Poll Failure" will be the text that will appear in the Description column name of the event record broadcast to AMOS when a poll fail condition occurs. If a threshold condition or poll restore condition occurs then the original text of "sysUpTime has been exceeded" will appear in the Description column name of the event record broadcast to AMOS. In the same way as the other components of Cisco Mobile Wireless Fault Mediator, **eval** statements (see Appendix B, "The Eval Statement") can be used within the event override lists.

*Table 4-3 The Attributes Available for Overriding Existing Event Column Names*

| Attribute | Description |
|---|---|
| Thresholdlist | Specifies column names to override if the threshold condition evaluates true. If a column name is specified within the Thresholdlist then it will override any column name contained in the event record which was previously defined in the event construction list. If the Thresholdlist is empty then the column names in the event construction list will be sent in the event record. |
| Pollfaillist | Specifies column names to override in a similar manner to the Thresholdlist. When a Polling Agent has been unable to contact a device, i.e., the device is unreachable, the Pollfaillist will be activated. If a column name is specified within the Pollfaillist then it will override any column name contained in the event record which was previously defined in the event construction list. If the Pollfaillist is empty then the column names in the event construction list will be sent in the event record. |
| Restorelist | Specifies column names to override when a poll restore condition occurs, i.e., when a device that was previously unreachable becomes contactable. If a column name is specified within the Restorelist then it will override any column name contained in the event record which was previously defined in the event construction list. If the Restorelist is empty then the column names in the event construction list will be sent in the event record. |

# Detailed Analysis of the Polling Process

This section consolidates the information already given to you by analyzing in detail the polling process for each of the four standard Polling Agents, and what their respective poll strategy attributes and poll outcomes are. In each section we examine a common scenario, how the Polling Agents function in each scenario and the possible poll outcomes.

## Polling for Presence—Using the Ping Agent

Polling for presence is used to ensure that a device is still present, live and contactable in the network by sending a packet of information—on a periodic basis—to an IP address and waiting for a response. Polling for presence is associated with the Ping Agent and the ICMP protocol. Polls for presence are defined using the Frequency and Timeout attributes (see Table 4-1 on page 4-5) and an internal test flag as a restore mechanism.

There are three possible outcomes from a poll for presence:

- Poll success—device contactable within the timeout period
- Poll fail condition—device unreachable within the timeout period
- Poll restore condition—device contactable after previously being unreachable

Each of these is now explained on the basis that the Polling Agent used is the Ping Agent.

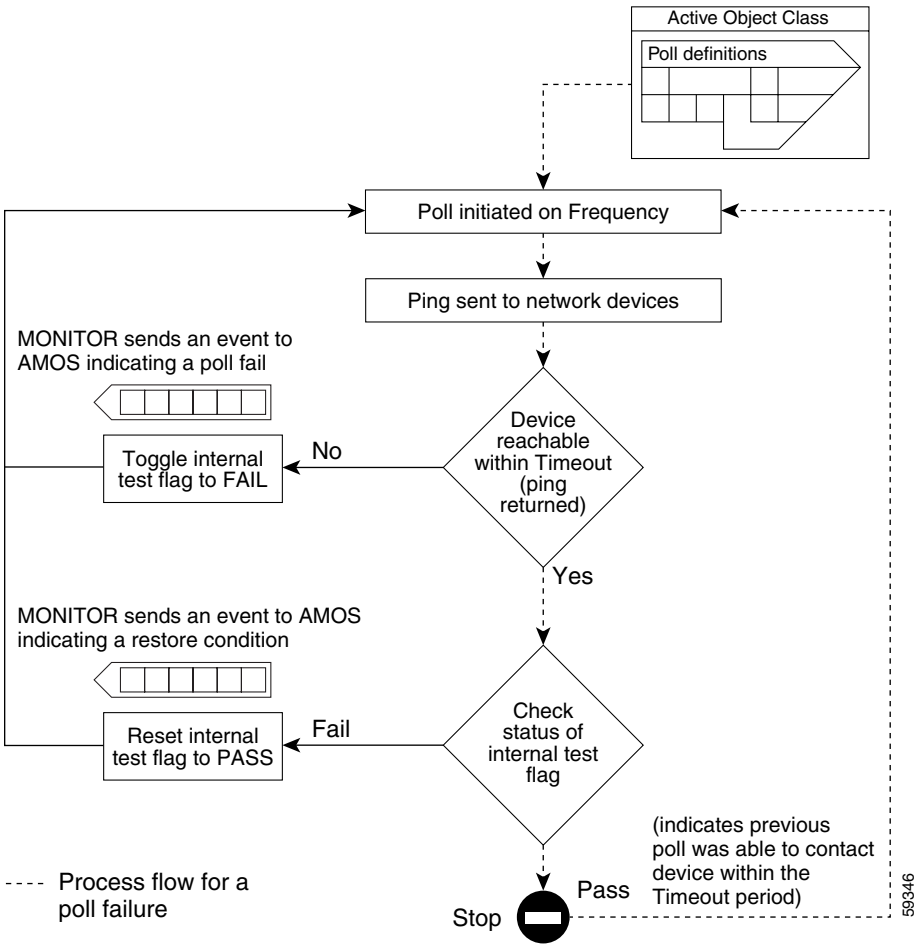
## Poll success

Scenario 1—All devices in the network are live and contactable. In addition the devices were contactable on previous polling attempts.

1. The ping poll is initiated based on the Frequency attribute of the poll strategy list.
2. A ping is sent to the appropriate network device.
3. The device sends a response within the Timeout period, i.e., the poll succeeds.
4. A check is made against the internal test flag. This is to determine whether or not the device was reachable the last time it was polled. Since it was (the flag will be set to pass), the poll will stop.
5. The poll will recommence when next initiated by the Frequency attribute.

In this case the poll is successful, the device contactable, and no event is sent to AMOS. Events are not generated when a poll is successful—only when a problem arises is the user informed.

***Figure 4-1 Process flow when the Ping Agent successfully polls a device***

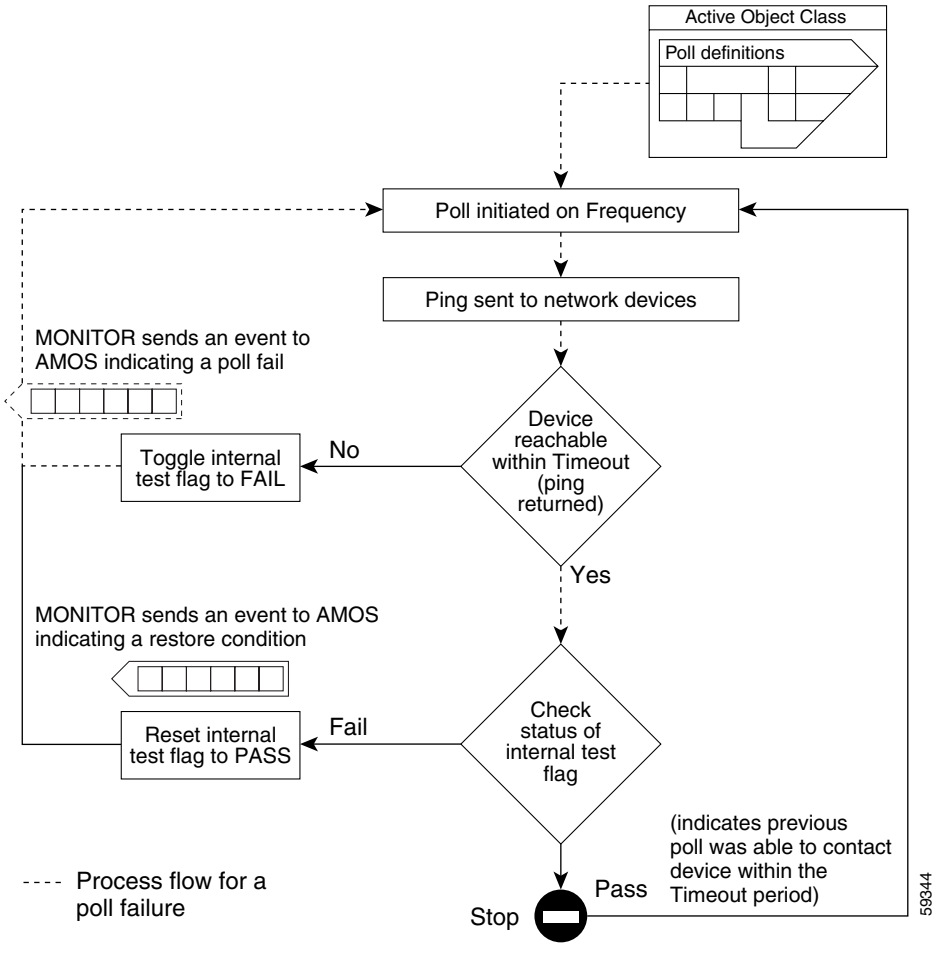


## Poll failure

Scenario 2—A device to be polled has been disconnected from the network, i.e., it is no longer contactable.

1. The ping poll is initiated based on the Frequency attribute of the poll strategy list.
2. The Ping Polling Agent attempts to ping the device; since it is not contactable, the device will fail to respond within the Timeout period.
3. The poll has failed, so an event will be generated and sent to AMOS with an appropriate name assigned to the EventName column name. In addition, the Pollfaillist will be examined to determine whether column names specified in the base generated event need to be overridden.
4. The internal flag test will be set to fail to acknowledge the fact that the device could not be contacted; finally, the poll stops.
5. As before, the device will be re-polled when next initiated by the Frequency attribute.

***Figure 4-2 Process flow when a device fails to respond to the Ping Agent***



A poll fail can occur for a multitude of reasons: for instance, the device may have been removed or disconnected from the network, or the traffic on the network may have been extremely high for a short period of time, preventing the ping packet being sent or retrieved from the host device. In a real-world scenario it is unlikely that a single pingFail would require human attention; if it were, the contents of a "typical" poll fail event would be similar to that shown below in Table 4-4.

*Table 4-4 The Column Names of a "typical" poll fail event*

| Column Name | Value |
|---|---|
| EventId | 1227 |
| EntityName | Device F |
| ClassName | Router |
| Description | Device has ping failed |
| EventName | pingFail |
| EventType | Event |
| Severity | Major |
| Contact | Jason Saunders |
| AssignedTo | Lance Armstrong |
| Acknowledged | Unacknowledged |
| Location | First floor communications rooms |
| CorrelatedId | 1225, 1226 |
| EventGroupID | 1227 |
| ActionGlyph | |
| CreateTime | 05:30 16 MAR 2001 |
| ChangeTime | 05:30 16 MAR 2001 |
| Occurred | 1 |
| CauseType | Symptom |
| ActionType | New |
| InternalAction | Assign |
| AgentAddress | 192.168.123.146 |

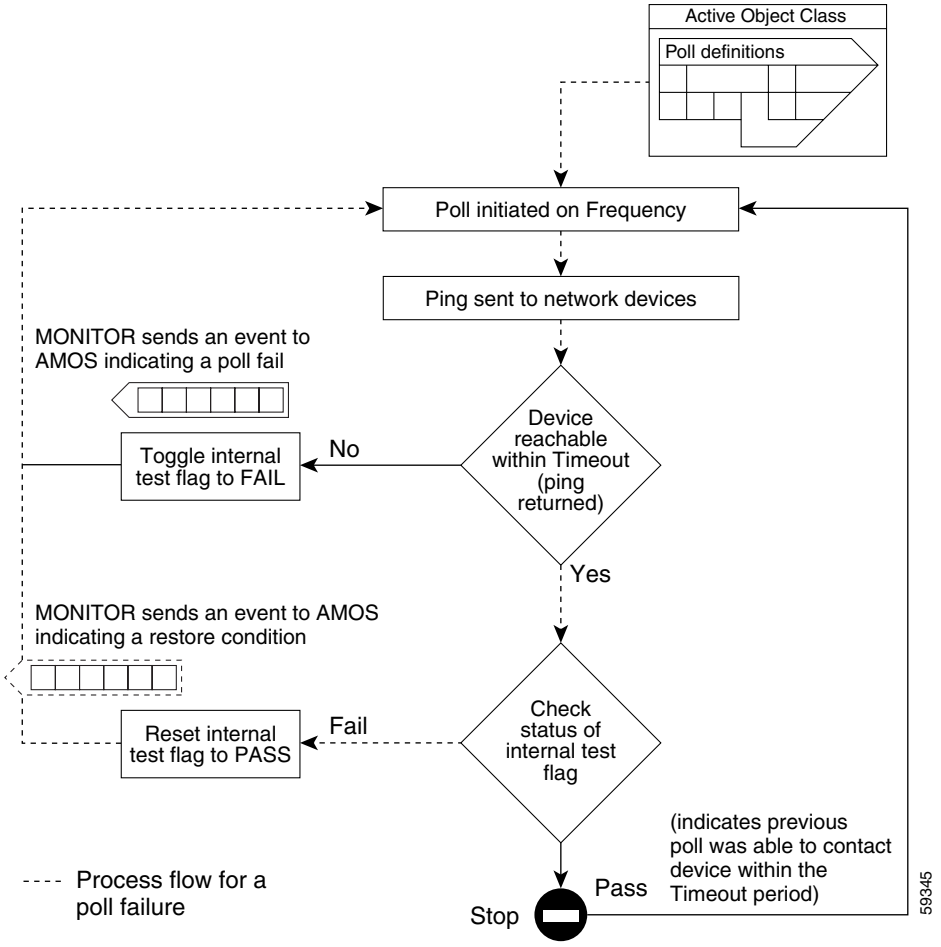
## Poll Restore

Scenario 3—A device previously unreachable (on the last ping attempt), becomes reachable again. This is a typical scenario immediately after a poll fail has occurred.

1. The ping poll is initiated based on the Frequency attribute of the poll strategy list.
2. A ping is sent to the appropriate network device.
3. The device sends a response within the Timeout period (since it is now reachable again), i.e., the poll succeeds.

4. The status of the internal test flag is checked; since the poll failed on the last occasion the internal test flag will be set to fail.
5. A restore condition is generated. The restorelist will be examined to determine whether column names specified in the base generated event need to be overridden. An event will be sent to AMOS indicating the device is now reachable and the internal test flag reset to pass.

***Figure 4-3 Process flow when the Ping Agent reaches a previously unreachable device***



# Polling for Details—using the SNMP Polling Agent

Polling for details is used to acquire MIB-related information from particular network devices. It is typically associated with the SNMP protocol and, hence, the SNMP Polling Agent. Polls for details are usually defined using the Threshold, Frequency, and Timeout attributes (see Table 4-1) and an internal test flag as a restore mechanism.

There are four possible outcomes from a poll for details (SNMP poll):

1. Poll success—device contactable within timeout period, threshold condition evaluates false.
2. Threshold condition—device contactable within timeout period, threshold condition succeeds, i.e., evaluates true.
3. Poll fail condition—device unreachable within timeout period, threshold condition is not evaluated.
4. Poll restore condition—device contactable after previously being unreachable, threshold condition does not succeed, i.e., evaluates false after previously being true.

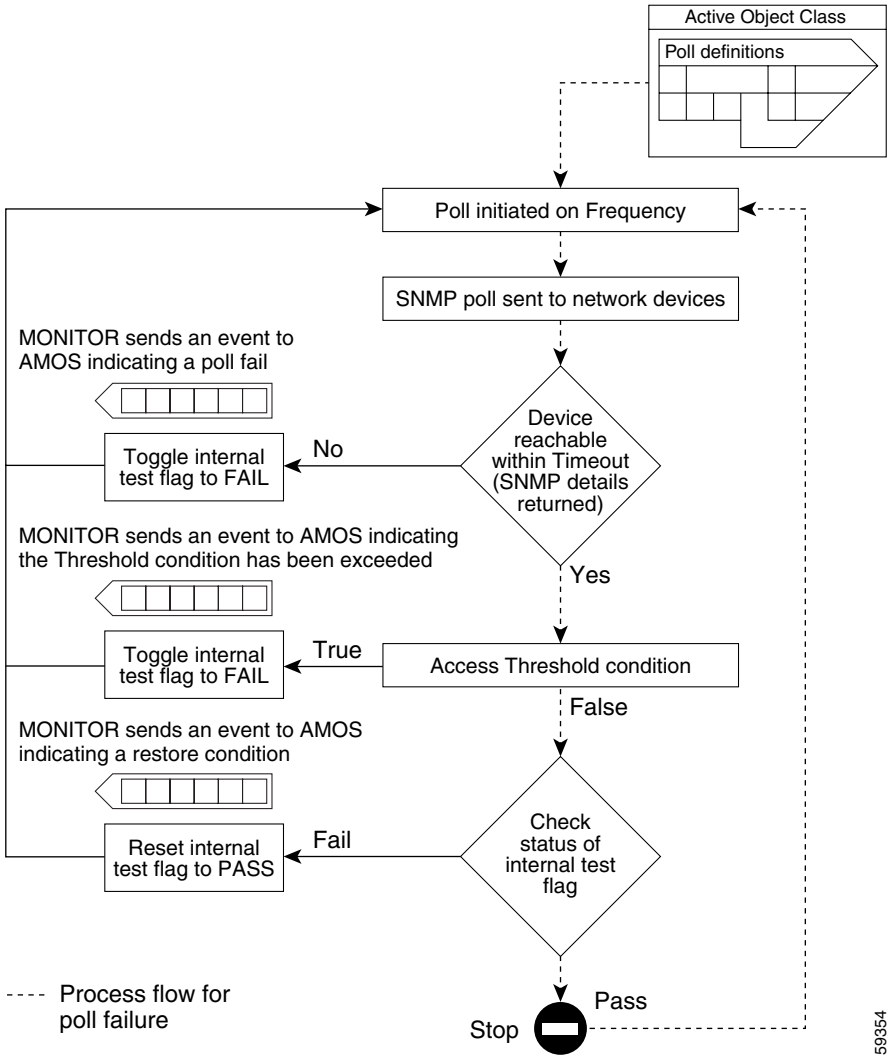
## Poll success

Scenario 1—All devices in the network are live and contactable. In addition, all the devices were contactable on previous polling attempts. The Threshold is defined as sysUpTime < 12,000 (i.e., 120 secs, since the MIB variable sysUpTime is defined in hundredths of a second since the object was last initialized).

1. The SNMP poll is initiated based on the Frequency attribute of the poll strategy list.
2. The Threshold (condition) attribute is used to specify the MIB information the SNMP Polling Agent is to retrieve from the network device.
3. The Agent successfully makes contact with the device within the Timeout period and retrieves the necessary MIB variable sysUpTime.
4. The Agent then assesses the logical aspect of the Threshold condition; for instance, sysUpTime < 12,000. If the actual value of sysUpTime is 15,000, the Threshold condition evaluates as false.
5. The state of an internal test flag is assessed; if the state indicates the device was reachable on the last polling attempt (which it was) the internal test flag will be set to true, the poll will stop and no event will be sent to AMOS.
6. The device will be re-polled when next required by the Frequency attribute.

In this case the poll is said to have been successful, the device contactable (within the Timeout period) and the Threshold condition did not succeed.

*Figure 4-4 Process flow when the SNMP Agent successfully retrieves details from a device*



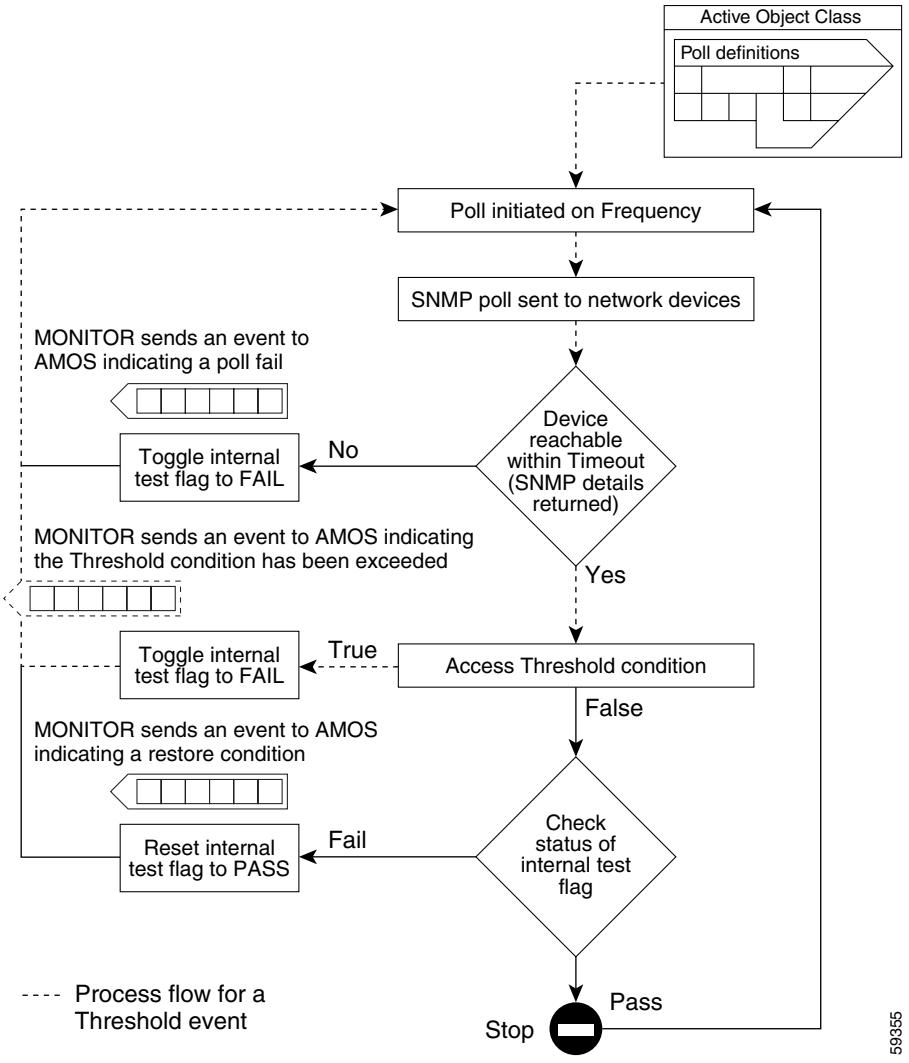

## Threshold event

Scenario 2—All devices in the network are live and contactable. In addition, the devices have also been contactable on previous polling attempts. The Threshold is defined as sysUpTime < 12,000.

1. The SNMP poll is initiated based on the Frequency attribute of the poll strategy list.
2. The Threshold (condition) attribute is used to specify the MIB variable sysUpTime the SNMP Polling Agent is to retrieve from the network device.
3. The Agent successfully makes contact with the device within the Timeout period and retrieves the necessary MIB information.
4. The Agent then assesses the logical aspect of the Threshold. If the actual value of sysUpTime is 10,000, the Threshold condition evaluates as true.
5. Since the Threshold evaluates as true the internal test flag will be set to fail.

6. The thresholdlist will be examined to determine whether column names specified in the base generated event need to be overridden. An event will be sent to AMOS indicating the Threshold condition has been exceeded.
7. The device will be re-polled when next required by the Frequency attribute.

***Figure 4-5 Process flow when the details retrieved by the SNMP Agent evaluate the Threshold condition as true***



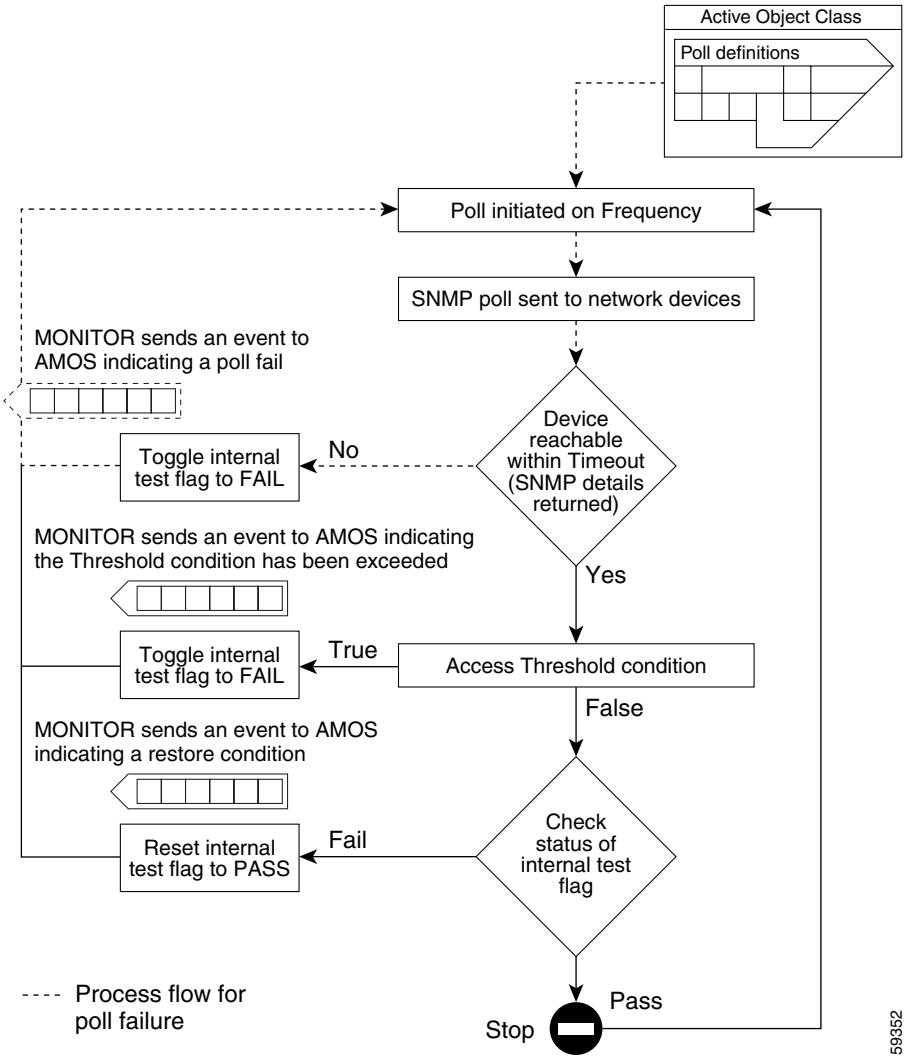
## Poll failure

Scenario 3—This situation is almost identical to the poll failure scenario in polling for presence. A device to be polled has been disconnected from the network, i.e., it is no longer contactable. The Threshold condition is defined as sysUpTime < 12,000.

1. The SNMP poll is initiated based on the Frequency attribute of the poll strategy list.
2. The SNMP Polling Agent attempts to retrieve the MIB variable sysUpTime from the network device (as specified via the Threshold attribute for the SNMP Polling Agent in the poll strategy list). Since the device is not contactable, the device will fail to respond within the Timeout period.
3. The poll has failed, so an event will be generated and sent to AMOS with an appropriate name assigned in the EventName column name. In addition, the pollfaillist will be examined to determine whether column names specified in the base generated event need to be overridden.
4. The internal flag test will be set to fail to acknowledge the fact that the device could not be contacted; finally, the poll stops.
5. As before the device will next be re-polled based on the period specified by the Frequency attribute.

**Note** On this occasion, the logical aspect of the Threshold condition is not evaluated since the poll and you have done has failed before it can be assessed.

*Figure 4-6 Process flow when a device fails to respond to the SNMP Agent*




## Poll restore

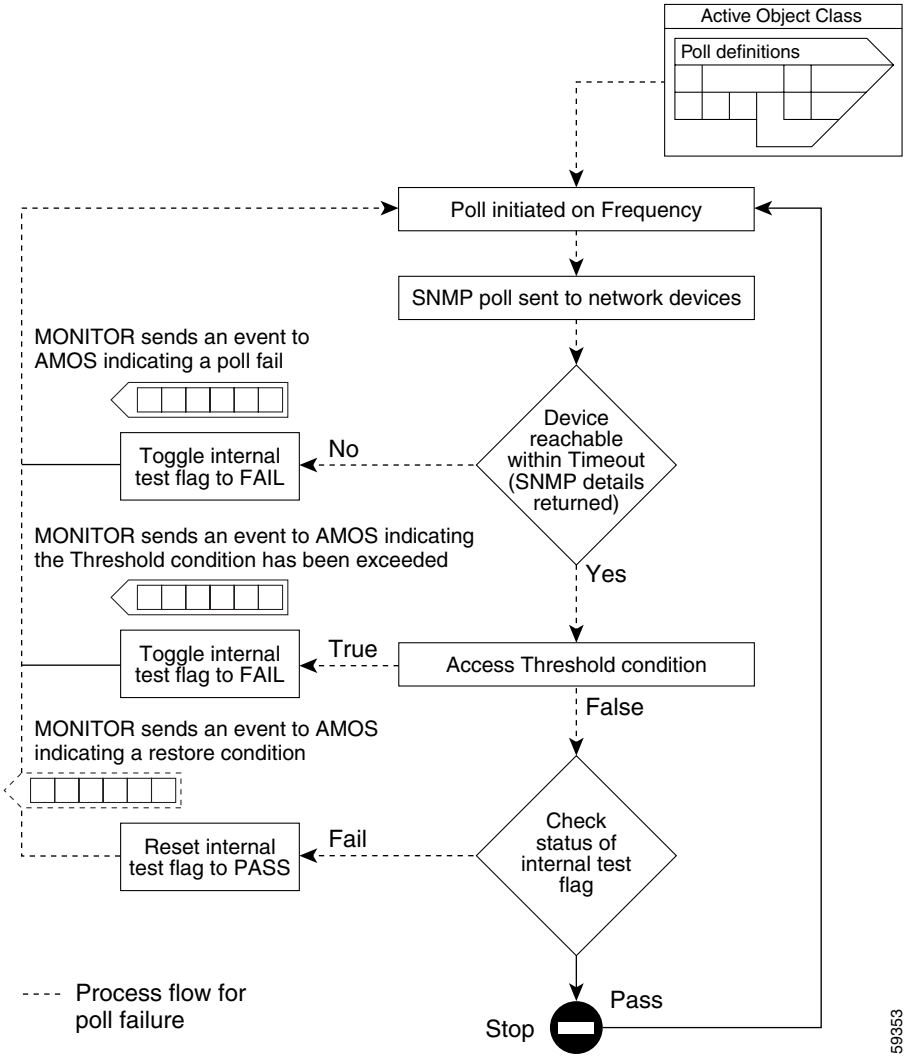
Scenario 4—This situation is almost identical to the poll restore scenario in polling for presence and is a typical scenario immediately after a poll fail condition. A device unreachable on the last SNMP poll attempt has become reachable again. The Threshold is defined as sysUpTime < 12,000.

1. The SNMP poll is initiated based on the Frequency attribute, as before.
2. The Threshold (condition) attribute is used to specify the MIB variable sysUpTime that the SNMP Polling Agent is to retrieve from the network device.
3. The Agent successfully makes contact with the device within the Timeout period and retrieves the necessary MIB information.
4. The Agent then assesses the logical aspect of the Threshold condition. If the actual value of sysUpTime is 15,000, the Threshold condition evaluates as false.
5. The state of the internal test flag is assessed; if the state indicates the device was unreachable on the last polling attempt (which it was) the internal test flag will be set to fail.

6. A restore condition is generated. The restorelist will be examined to determine whether column names specified in the base generated event need to be overridden. An event will be sent to AMOS indicating the device is now reachable and the internal test flag will be reset to pass.
7. As before the device will be re-polled based on the period specified by the Frequency attribute.

MONITOR also generates a poll restore event when the Threshold condition evaluates false after being evaluated true the last time the device was reachable. Thus there are two scenarios for a poll restore with the SNMP Polling Agent; the first occurs as a result of a poll fail indicating the device was unreachable, the second as a result of a change in the Threshold condition from true to false.

***Figure 4-7 Process flow when the SNMP Agent reaches a previously unreachable device or when the Threshold changes from true to false***



# Polling for traps—using the Trap Polling Agent

Traps are asynchronous notifications that enable an Agent to report a condition to a management station. Functionally, Trap Polling Agents have the simplest mechanism of all the Polling Agents, as they are reactive. There is no Timeout attribute; triggering simply occurs when the trap is received by the Trap Polling Agent. While there is no actual Threshold test in the form used when polling for details ("Polling for Details—using the SNMP Polling Agent" section on page 4-13) the Threshold condition is used to specify the type of trap and MIB information the Trap Polling Agent listens for. The fields you can specify in the Threshold attribute are any values in the trap MIB definition as well as the fields shown in Table 4-5.

***Table 4-5 Fields available in the Threshold attribute of a trap poll definition***

| Field | Description |
|---|---|
| TrapName | Name of the trap being processed. |
| TrapDescr | Description of the trap being processed. |
| TrapNo | Reference to the trap type value of the trap being processed, see Table 4-7. |
| TrapSpecNo | Used only when the enterpriseSpecific trap is used (i.e., trap type value and TrapNo (see above) equal 6). The TrapSpecNo identifies which trap type, of potentially many, this refers to. |

By default, you can also include logical operators to further restrict the conditional statement. A typical trap condition is shown in Table 4-6.

***Table 4-6 Defining a trap for MIB variables "linkUp" and "ifindex"***

| Attribute | Value |
|---|---|
| Threshold | `"TrapName = 'linkDown' AND ifIndex = 3"` |

There are six generic traps (plus an enterprise-specific trap) defined in RFC 1215, maintained by the Internet SOCiety (ISOC), that are commonly used with the SNMP protocol; these are listed in Table 4-7.

***Table 4-7 Trap type values, names and descriptions***

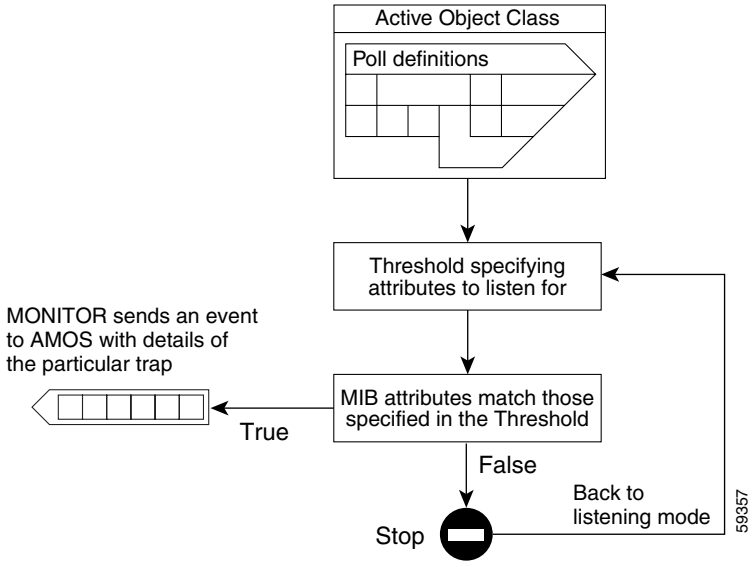
| Type Value | TrapName | Description |
|---|---|---|
| 0 | coldStart | Signifies the sending device is reinitializing itself and may have been altered. |
| 1 | warmStart | Signifies the sending device is reinitializing itself and has not been altered. |
| 2 | linkDown | Signifies the sending device recognizes the failure of a communication link. |
| 3 | linkUp | Generated when a recognized communication link comes up or gets restored. |
| 4 | authenticationFailure | Generated when the sending device receives a message that is not properly authenticated, e.g., an incorrect login attempt. |

*Table 4-7 Trap type values, names and descriptions (continued)*

| Type Value | TrapName | Description |
|---|---|---|
| 5 | egpNeighborloss | Signifies that an Exterior Gateway Protocol (EGP) neighbor for whom the sending device was an EGP peer has been marked down and the relationship no longer exists. |
| 6 | enterpriseSpecific | Signifies the sending device recognizes some enterprise-specific event has occurred, i.e., it is not one of the six standard traps. |

The Trap Agent listens to the network; when the information from a device meets the criteria defined in the Threshold attribute of the poll definition (i.e., the information the Trap Poll is looking for), it triggers an event which MONITOR sends to AMOS.

*Figure 4-8 Process flow for the Trap Agent*




While the only outcome of a Trap Poll is the generation of an event, the content of the event record sent to AMOS will depend on the trap and fields specified in the Threshold condition. For instance, the information contained in the event record for a coldStart trap will be different to that specified for a linkUp trap.

The number of TrapNames and fields available in the Threshold condition is unlimited, since you can configure new fields unique to your requirements. For example, you may wish to utilize the environmental characteristics available on a Cisco high series router such as a 7500 to specify a poll for a trap associated with the inlet air temperature to the processor. In this case, a new TrapName, "CiscoInletTempTrap" would be defined. The router would then be configured to send the trap CiscoInletTempTrap when the inlet air temperature to the processor exceeded the criteria defined during the router configuration process (for instance, 55 degrees Celsius). In this case the Threshold attribute in the poll definition would appear as shown in Table 4-8.

*Table 4-8 Defining a trap using the environmental characteristics of a router*

| Attribute | Value |
|---|---|
| Threshold | `"TrapName = 'CiscoInletTempTrap'"` |

# Polling for messages—using the Syslog Polling Agent

Polling for messages is used to "watch" for alterations in the Syslog files by parsing them. Syslogs allow a device to deliver messages to another device; they have a particular format that associates a service facility, severity and priority with the messages generated. Like polling for traps, polling for messages only has one plausible outcome; when the Threshold condition is met, an event with column names associated with the Syslog will be sent to AMOS by MONITOR. When polling for messages, the Threshold condition operates in the same manner as when polling for traps—fields are included in the Threshold statement. These fields are identified in Table 4-9.

*Table 4-9 Default fields available in the Syslog Threshold condition*

| Field | Description |
|---|---|
| Date | Date information. |
| Time | Time information. |
| NodeName | A network node name, device name or entity name. |
| Service | The type of service facility associated with the device. For instance a mail delivery system, user process, system process, etc. |
| Message | A text string contained in the parsed file. Typically, Syslog files contain many strings of free-form text so it is imperative for the parser to be able to filter on a string or text wild card. |

An example of a valid Syslog Threshold condition is shown in Table 4-10.

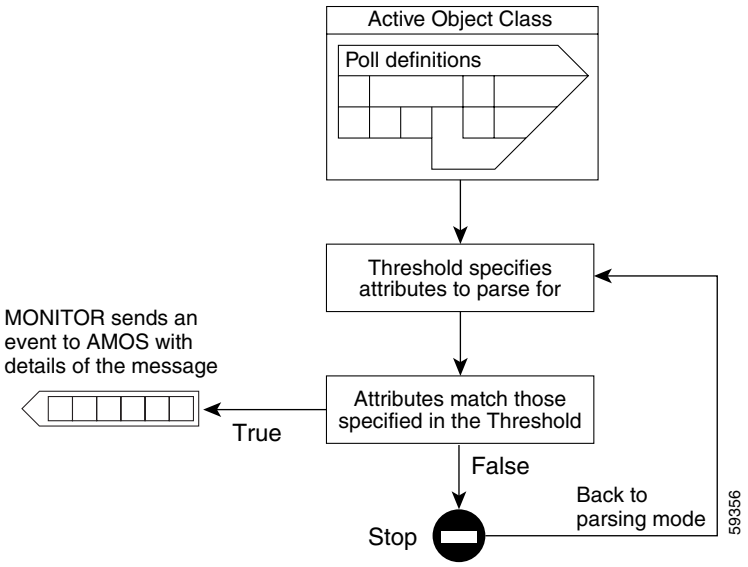
*Table 4-10 Defining a Syslog Threshold condition*

| Attribute | Value |
|---|---|
| Threshold | `"NodeName like 'Ami' AND Message like 'login fail'"` |

In this situation, the Syslog Polling Agent would be parsing Syslog files, filtering for devices which contained the text string “Ami” indicating a login attempt had failed. If the central server is called “Amidala”, then a Syslog event will be sent to AMOS if the Syslog Polling Agent parses a Syslog file from Amidala with one or more failed login attempts. Under normal circumstances, login failures may not be of significance or even warrant the attention of the user—unless Amidala is a central server essential to the operation of a inventory management system or an enterprise resource planning package, and therefore a critical device. In this situation, Syslog files generated by repeated authentication failures, indicating a possible breach of security, would be cause for concern. In this instance, the events generated by the Syslog Agent could be correlated into an alert by AMOS notifying the network administrator of these attempted, potentially destructive, actions. The stream of event records generated by the repeated logins may also provide additional information indicating the location of the terminals and the times of the authentication attempts which could assist the network administrator in apprehending the culprit.

Like the Trap Agent, the Syslog Agent is configurable; additional user-defined fields can be included depending on your system requirements.

***Figure 4-9 Process flow for the Syslog Agent***




# Conclusion

This chapter has described all aspects of the polling process; firstly, by providing an overview of MONITOR, the polling process controller; secondly, by looking at the expected outcomes of the polling process—the events; thirdly, by describing the types and classifications of Polling Agents MONITOR uses to retrieve information from the network; fourthly, by outlining the attributes contained in the poll definitions (the first of the AOC extensions) along with a detailed explanation of the function of each attribute; finally, by describing the outcomes of the poll definition for each type of Polling Agent in various scenarios.

As a result of the polling process there is a multitude of “raw” events in the event stream—each associated with a particular instance on the network. The next step is to make distinctions between the events and deduce their relative importance. Are there any relationships between the events? Are multiple events coming from one area of the network indicating a potential problem? Which events are merely symptoms of a greater cause?

The answer to these questions comes from AMOS, the event correlation engine, which correlates (and processes) the event stream received from MONITOR. The function of AMOS is controlled by the event correlation methods, which are the topic of the next chapter.

CHAPTER 5

# The Event Management Process

This chapter details the event correlation methods, the second of the AOC extensions from Cisco Mobile Wireless Fault Mediator. It examines the complete structure of an event correlation method and details each component and the attributes used to construct a method. Finally, there is a preview of the role played by the method conclusions, the final stage of the fault management process.

## Managing events

The goal of event management within Cisco Mobile Wireless Fault Mediator is to reduce the amount of superfluous network status information presented to the user. This takes the form of correlating certain events in the event stream together to arrive at a single root cause, suppressing events, changing their severity under certain circumstances, deleting events that are no longer relevant, or even suspending polling to particular devices when certain events are received from them.

This process involves several key steps: storing events, filtering events, analyzing relationships between events on the network and the network topology and, finally, taking action based on the correlation process. Each of these areas is now discussed before introducing the event correlation methods that define how AMOS, the correlation engine of Cisco Mobile Wireless Fault Mediator, functions.

### Event storage

Each event that transpires on the network must be logged or stored somewhere for later use and for AMOS this is the AMOS Events database. The AMOS Events database contains the current set of events and alerts. Each record contains a series of column names which are used by AMOS to determine what action needs to be undertaken on each event. The event records are well-structured to ensure event records are handled consistently. The column names of an event record and the AMOS Events database are thoroughly explored in Appendix A, “AMOS Databases.”

## Event filtering

The second step is to determine whether an incoming event satisfies a conditional test. The nature of the test is described in further detail in the section on the event correlation methods later in this chapter.

## Correlation and topology considerations

For all events that pass the filtering process, the third step is to undertake a correlation process. This process involves comparing the column names of the incoming event to the network topology model, Containment Model and other events that have transpired on the network to establish whether there is a relationship between events; for instance, did the failure of one device cause a series of events on devices "downstream" of the failed device? Or are there other implications for devices connected to the failed device?

## Taking action

The final step is to action any faults that have been identified as a result of the correlation stage. Should events on other network devices be suppressed or deleted as a consequence of an event? What database modifications inserts, updates and deletes are required as a consequence or correlation? Are directives required to be run due to correlated results?

# The event correlation methods

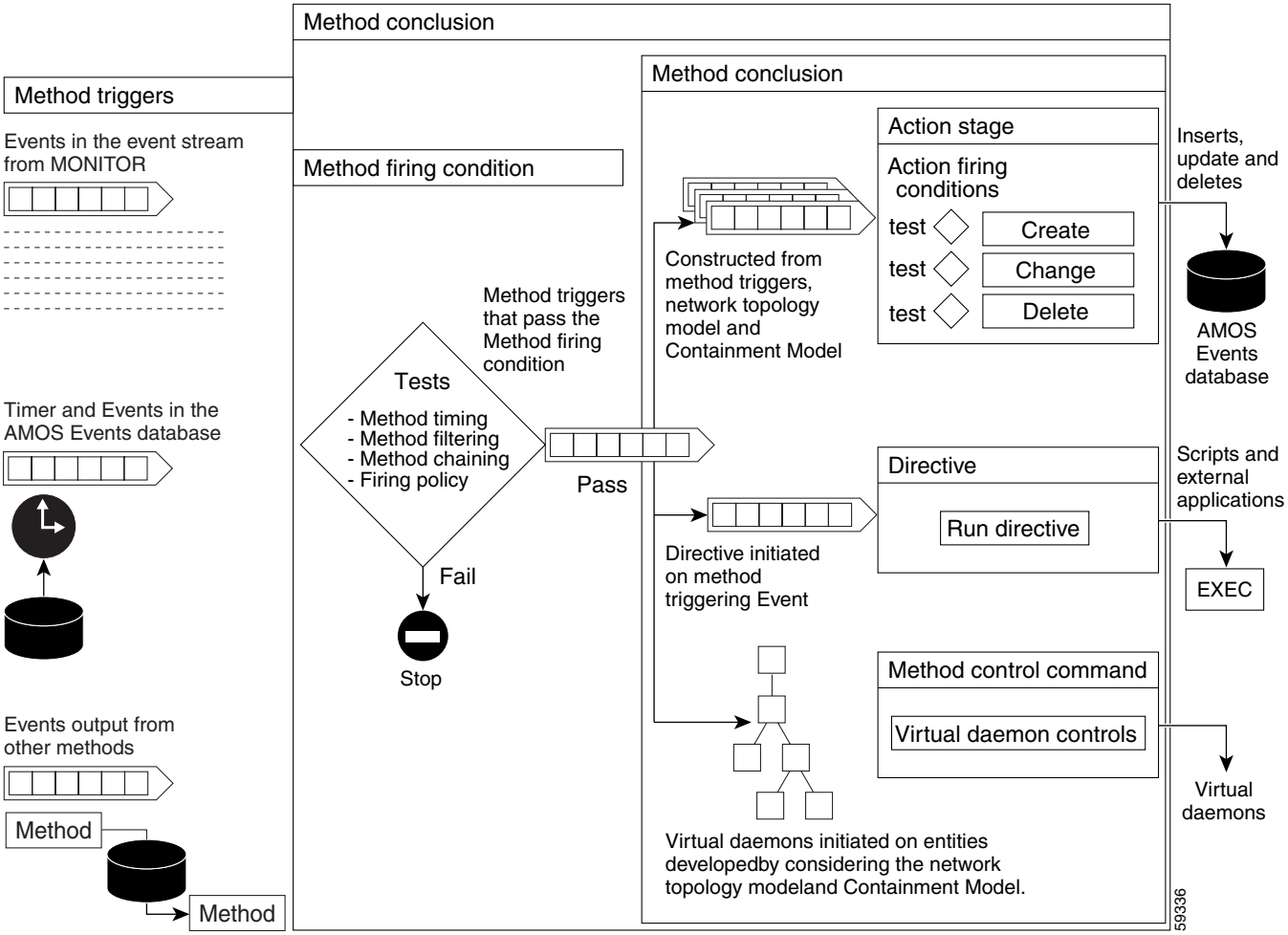
The operation of AMOS is determined by the event correlation methods, which are extensions in the AOCs; thus, each instance of an object instantly has event correlation methods associated with it. When AMOS is launched, it extracts (via the MWFM NMOS component CLASS) the event correlation methods from the relevant AOCs. The event correlation methods are then on standby until initiated by a trigger. The triggers are one of the four components of the Method Template:

1. The method name.
2. The method triggers.
3. The method firing condition.
4. The method conclusion.

**Note** Each event correlation method follows a standard template to fulfill the requirements outlined in the "Managing events" section on page 5-1.

Each method's template is constructed in the AOC browser (see Figure 3-2 on page 3-4).

*Figure 5-1 The components of an event correlation method*




# The method name

The method name is an attribute that declares the name of the current method. Any string of text is accepted as a valid entry and must be unique within the specified AOC. For example, “Create” and “Escalate” are valid method names.

A feature of both the event correlation methods and the poll definitions is the economy of description offered by the AOCs’ inheritance of policies. Classes lower in the hierarchy will inherit the functionality of all methods from their parent classes, unless the event correlation method is locally redefined to override the function of the inherited one. This is done by creating a new method in the child class with the same method name as the one derived from the parent class.

The method name is also used as a means of method control; it is the method name which is the basis for method chaining. For further information see the “Method chaining” section on page 5-6.

# The method triggers

An event correlation method can be initiated or "triggered" in any of three ways:

1. By events in the event stream broadcast from MONITOR—a non-timed method.
2. By a timer defined in the event correlation method itself—a timed method.
3. As a result of applying method chaining—a non-timed method. The method can be triggered by events output from another method or by another method firing (or even after it not firing).

The notion of triggering introduces two associated terms hinted at above, timed and non-timed methods. Both are components of the method firing condition.

# The method firing condition

The method firing condition is quite simply a series of conditional tests to determine whether the method conclusion is fired or not. If the conditional tests of the method firing condition are passed, then the method will be fired. In such a case, processing will continue and the method conclusion will be run (activated). If the tests in the method firing conditions are not passed, processing is terminated. The method is not fired and the method conclusion is not run.

**Note** Firing, in the context of an event correlation method, is when the method firing condition is successfully evaluated, and processing continues to the method conclusion, where some action takes place.

The firing condition can be broken down into four key components:

- The method timing section
- The method filtering section
- The method chaining section
- The firing policy

These components are discussed below.

# The method timing section

The first component of the firing condition relates to the triggers outlined in the previous section. As specified, there are three triggers. These triggers, and thus the operation of the event correlation methods, can be further categorized into timed and non-timed methods.

## Timed event correlation methods

A timed method is one that is triggered by specifying a time in seconds, minutes, hours or days in the relevant field in the AOC browser. For example, specifying a time of five minutes will mean the event correlation method will be triggered every five minutes and run on any event present in the AMOS Events database that satisfies the remainder of the conditional tests in the method firing condition.

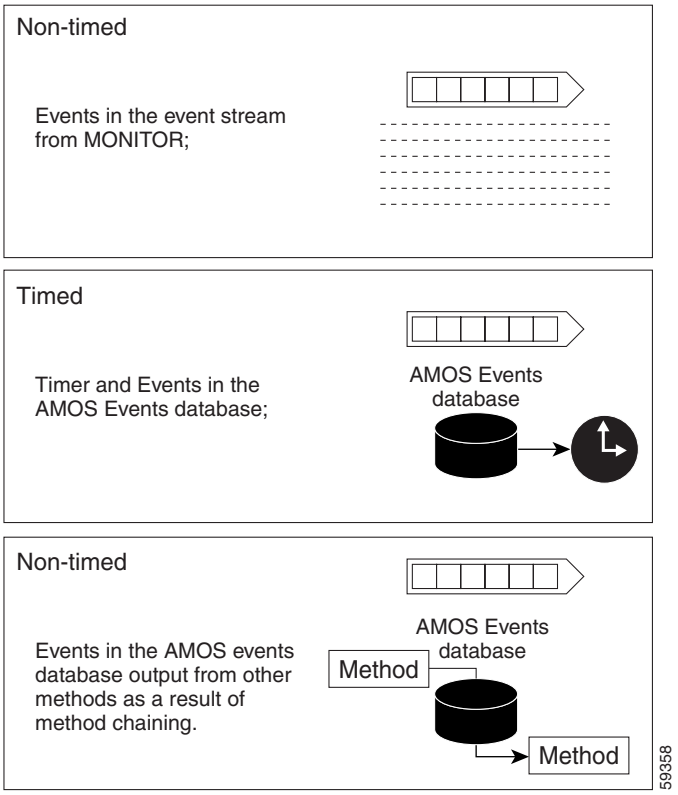
## Non-timed event correlation methods

A non-timed method is one that is triggered on the basis of an interrupt from an external source. There are two external sources:

1. Events in the event stream broadcast from MONITOR.
2. As a result of applying method chaining. In such cases the source is events output from another method or events in the AMOS Events database.

In both cases, the conditional tests in the method firing condition will be run on the event that triggered the event correlation method, whether that is the event(s) in the event stream or the event(s) output from another method.

***Figure 5-2 The three triggers' input sources***




# Event and entity filtering

The next component of the method firing condition is the conditional test. Cisco Mobile Wireless Fault Mediator uses two means of filtering in the firing condition, the event filter and the entity filter. The event filter allows you to filter on the column names of an event record contained within the AMOS Events database, while the entity filter allows you to set criteria based on column names of entities in the AMOS Entity database.

## The event filter

The event filter uses a series of logical operators and the column names of the AMOS Events database to constrain the execution of an event correlation method. For example, you may choose to define a method called “routerPingFail” which runs on ping fail events from routers when the Severity of the event is greater than “Unknown” (e.g., Minor, Major, Critical - see Table A-1 on page A-1, for the precedence of the values in the Severity column name of an event record). The event filter for such a condition would be defined as shown in Table 5-1.

*Table 5-1 Specifying the event filter*

| Attribute | Value |
|---|---|
| event filter | `"EventName='pingFail' AND Severity > eval(int,'$UNKNOWN')"` |

You can see how the column names of an event record can be used in a filter as well as some OQL syntax, namely the **eval** statement. EventName and Severity are column names of an event record (as tabulated and described in Appendix A, “AMOS Databases.” The **eval** statement enables the column names of an event record to be accessed and evaluated. The syntax and functionality of the **eval** statement is comprehensively explored in Appendix B, “The Eval Statement.”

In the example, the event filter portion of the firing condition would be passed if the incoming event record was a pingFail with a Severity of greater than Unknown.

## The entity filter

The second filtering condition, the entity filter, operates in the same manner as the event filter except it filters against the network topology constraining the execution of an event correlation method using column names in the AMOS Entity database and a series of logical operators. The column names are outlined in Table A-2 on page A-3.

For instance, you may choose to limit the method to run only on events from a particular part of the network or subnet. In this case the entity filter would be defined as shown in Table 5-2.

*Table 5-2 Specifying the event filter*

| Attribute | Value |
|---|---|
| entity filter | `"Address = eval(list type text, this -> Address)"` |

# Method chaining

Method chaining allows a series of simple methods to be combined to form more complex methods; output from one method can be utilized and processed by a second method after the first has carried out its processing. This improves efficiency in determining root causes and streamlines the number of methods required in the AOCs.

Method chaining is achieved by linking event correlation methods via the Event Method dialog in the AOC browser. The process of linking methods in the Event Method dialog injects a series of entries into the method firing condition of the event correlation methods. These entries not only link the event correlation methods but also determine the sequencing of the methods; there are four method chaining possibilities. These are described in Table 5-3.

*Table 5-3 The four method chaining possibilities*

| Chaining method | Description |
|---|---|
| Evaluate after | Run the present method on the AMOS Events database after a previous method has been evaluated, regardless of whether it fires or not. |
| Evaluate after fired | Run the present method on the AMOS Events database after a previous method fires. |
| Evaluate after not fired | Run the present method on the AMOS Events database after a previous method does not fire. |
| Evaluate when fired | Run the present method on events output from another method when that method fires. |

# The firing policy

The firing policy is the final attribute that dictates whether the event correlation method conclusion will be fired or not. The firing policy is instigated after the event and entity filtering and method chaining has been assessed. There are three possible attributes available when specifying the firing policy:

- Time-to-live (Ttl)—an integer value that determines how many times the method is fired (or "actioned") before it is deactivated. For example, if the Ttl value is "3" the method will fire three times and then deactivate until an external process (such as input from the AOC browser) reactivates it. If a value of zero is specified the method will fire indefinitely.
- Number of fires before execute—an integer value that specifies the number of times the method firing condition must be evaluated successfully before the method conclusion is executed. If the specified integer has a value of "4", this means the method firing condition must be satisfied four times; on the fourth time, the method conclusion will run. Furthermore, once the method firing condition has been evaluated the number of times specified by the "number of fires before execute" attribute (in this case four), the method will be fired on each subsequent successful evaluation of the method firing condition. Thus, in this example, the method will be fired on the fifth successful evaluation, the sixth successful evaluation, etc. If a value is omitted or specified as zero, the method always fires.
- Repeat condition—an integer value that operates similar to the number of fires before execute condition, except it repeats after the method conclusion has been executed once. If the integer specified is "n", then the method conclusion will be fired after the n, 2n, 3n, etc. successful evaluations of the method firing condition. For instance, if a value of "3" is specified the method conclusion will be run after the third successful evaluation. The method conclusion will not be run again until the method firing condition has been successfully evaluated on three more occasions. Thus, the method will be run on the sixth successful evaluation, the ninth successful evaluation, etc.

# The method conclusion

Once the method firing condition has been passed, processing of the method conclusion commences. The method conclusion consists of two components:

- The actions stage—determines how the event records in the AMOS Events database are manipulated; for example, inserting, updating and deleting event records.
- The method control daemons—these enable the actions in the method conclusions to be run on devices specified by the control location without having to re-run the entire event correlation method.

Similar to the event correlation methods, each component of the method conclusion requires a trigger or a mechanism to initiate it. These are now discussed with respect to the event management process; you should move to Chapter 6, "The Method Conclusion In Action," for a detailed exploration of the method conclusion in action.

## Constructing actions triggers

Action triggers are a list of all events and alerts that have transpired on all devices (and entities contained within the devices). The actions triggers are effectively the input to the actions stage of an event correlation method conclusion.

The sole purpose of developing actions triggers is to generate a more comprehensive event and alert list which will be the input to the actions stage of the event correlation method conclusion. Should events on all devices be considered? Or just events on instances of devices? Is it necessary to consider events on all devices connected to a particular device? Or events on devices isolated by the failure of a discrete entity? Should we consider any or all of the events the entities contained in one of the devices just described? These are complex questions; however, it is possible to construct methods to cover any of the situations identified.

In order to construct the actions triggers it is necessary to establish exactly which entities (and therefore, which events from these entities) are to be used for the actions stage.

To facilitate this, the network topology model and the Containment Model described in Chapter 2, "Object-oriented Principles and MWFM NMOS," are used to enable events that transpire on entities contained within devices to be considered in the actions stage, even though these entities may not be instances of AOCs. The ability of Cisco Mobile Wireless Fault Mediator to combine the network topology model and the Containment Model is one of the keys to Cisco Mobile Wireless Fault Mediator's ability to accurately establish the root cause of network problems.

## Which devices do I run the event correlation method conclusion on?

The control location attribute is used to scope the method to particular network objects within the network topology model.

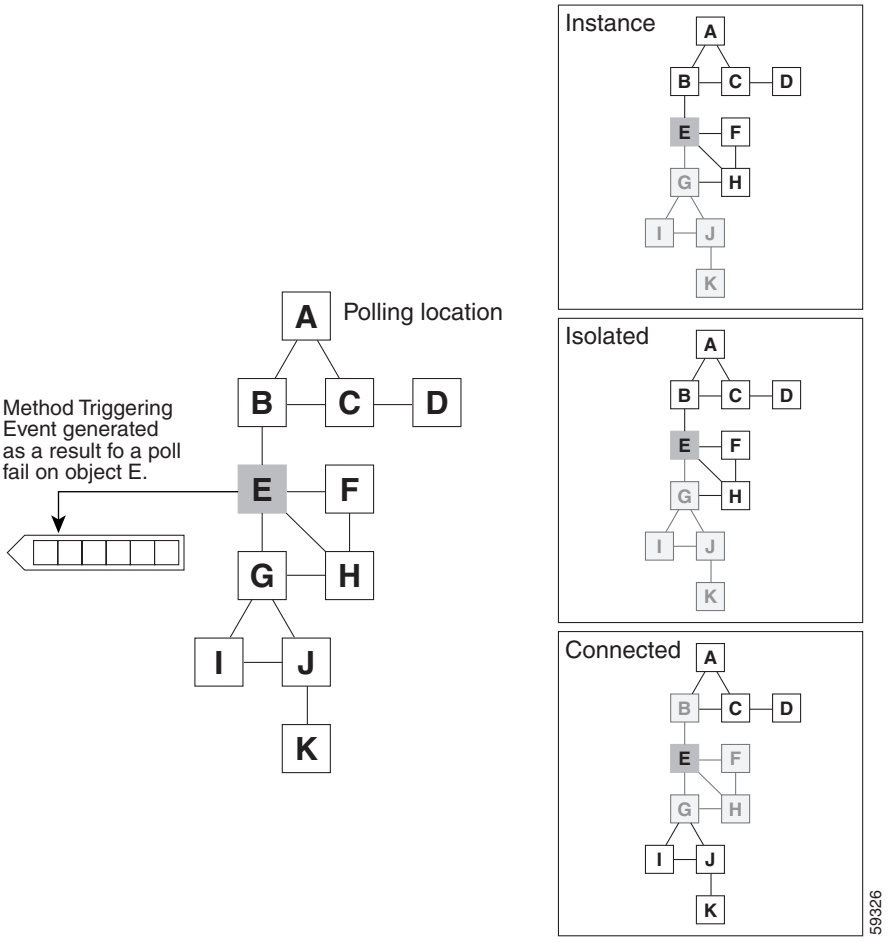
**Note** Scoping a method is determining which network objects in the topology model will have the event correlation method conclusion applied to them.

There are three control location possibilities, as described Table 5-4 with reference to the sample network in Figure 5-3:

***Table 5-4 Control location possibilities***

| Control location | Assigned value | Description |
|---|---|---|
| Instance | 0 | Selects only the object that generated the event which triggered the event method. In Figure 5-4, the instance that generated the original event is object E. The actions triggers will consist of all events that have transpired on that instance (subject to the application of the Containment Model—described in the next section). |
| Isolated | 1 | Selects all objects that can only be reached by going though the object which generated the event that triggered the event correlation method. In Figure 5-4, the objects that would be isolated by the failure of Object E are Objects G, I, J and K. H can still be reached via F. The actions triggers will consist of all events on isolated objects plus the method triggering event (subject to the application of the Containment Model—described in the next section). |
| Connected | 2 | Selects all objects that are directly connected to the object which generated the event that triggered the event method, Object E. In Figure 5-4, these objects are B, F, G, and H. The actions triggers will consist of all events on connected objects plus the method triggering event (subject to the application of the Containment Model—described in the next section). |

*Figure 5-3 Sample network demonstrating the application of the control location*




When a control location of isolated is specified, there is an additional requirement to relate the location of the Polling Agent to the network topology, called isolated from. This allows you to establish a direction or a relationship between the object where the event transpired in the network and where the object was polled from. In the sample network above, the polling location is Object A. We have previously identified the fact that the failure of Object E would cause a situation where Objects G, I, J and K are isolated from the network. This observation is only valid if Object A is the polling location. If the polling location was Object G, the entities isolated by the failure of Object E would be entirely different; they would be Objects A, B, C, D, F and H. Thus, it should be clear that when you want to identify certain objects that are isolated by the failure of another object you must clarify this by establishing a relationship between the object being polled and the polling location.

Specifying the polling location is achieved by extracting the IP address of the Polling Agent from the method triggering event. This can be done using an **eval** statement similar to the one shown in Table 5-5.

*Table 5-5 Specifying the Polling Agent location*

| Attribute | Value |
|---|---|
| isolated from | `"EntityName = eval(text,'&AgentAddress')"` |

In the context of our original example, the above statement would have extracted the IP address from the polling location and equated it with Object A.

## What level of containment do I consider?

Once you have established which objects (or devices) the event correlation method will consider events on, it is also necessary to specify the containment level you wish to allow, since objects can contain other entities. For instance, a switch has a chassis that contains a series of cards, which may in turn contain a series of ports; additionally, these ports may be associated with a series of VLANs. Specifying the target level of containment you wish to consider gives Cisco Mobile Wireless Fault Mediator information about how it is to recurse through the Containment Model resolved during the discovery process. The level of containment is established by using the run in container attribute which contains the target attribute and three additional arguments: traverseUp, traverseDown and controlFlag.

*Table 5-6 Establishing the level of containment*

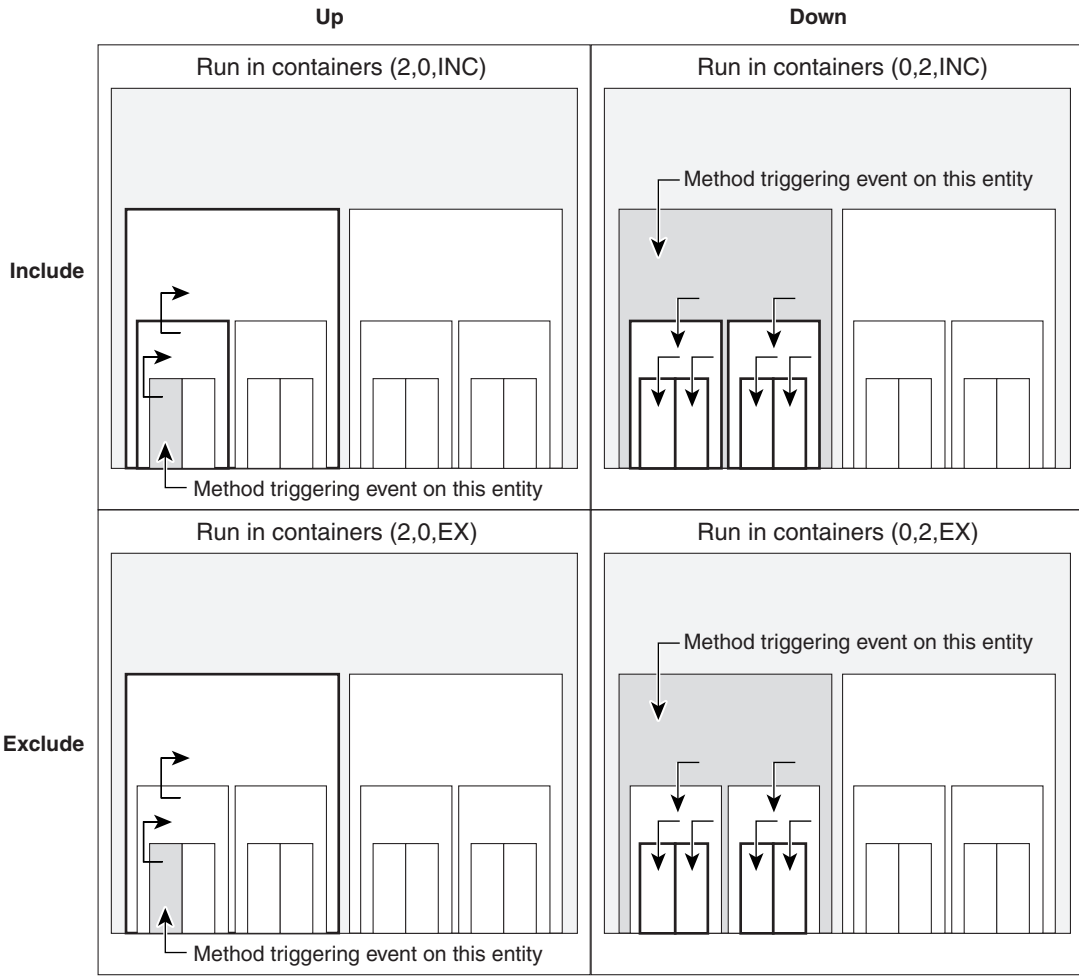
| Attribute | Value |
|---|---|
| Run in container | `target = (traverseUp, traverseDown, controlFlag)` |

- target—a list of all entities for inclusion in the generation of the actions triggers; events transpiring on the entities in the target list will be included in the actions triggers. The list is based on the three integer fields (traverseUp, traverseDown and controlFlag) which designate the containment level. A description of each integer follows.
- traverseUp—an integer value used to specify the number of levels to traverse up through the Containment Model. For instance, if an event associated with a card in a switch has occurred, it is possible to recurse up one level of the physical container and capture all events that have transpired on the switch. These events can then be incorporated in the actions triggers.
- traverseDown—an integer value used to specify the number of levels to traverse down through the Containment Model. Consider the same card in the switch above; it is possible to specify a traverseDown value that will allow you to consider all the events that have transpired on the ports of the card. When traversing down through the Containment Model, all paths are considered; for instance, when traversing from card level to port level it is not possible to discriminate between events occurring on different ports. Any event occurring on any port on the card in question will be included in the actions triggers.
- controlFlag—used to denote whether the event(s) on the entities encountered when traversing up and down the levels of the Containment Model should be included or excluded when compiling the "new" list of events and alerts which are the actions triggers for the actions stage. If controlFlag is set to includerecursed, then any event transpiring on an entity encountered when traversing up or down the Containment Model will be included in the actions triggers. If controlFlag is set to

excluderecursed, then any event transpiring on an entity encountered when traversing up and down the Containment Model will be excluded from the actions triggers; the only events considered will be those on the final destination entities.

Figure 5-4 demonstrates the uses of the run in containers command to recurse through the Containment Model. To disable recursing through the Containment Model the run in container attribute is simply left unassigned.

***Figure 5-4 Recursing through containers***




The output generated by applying the control location to the network topology model and considering the Containment Model is a list of EntityNames whose events and alerts will be considered in the actions stage of the method conclusions—these are the actions triggers.

The operation of the actions stage of the event correlation method conclusion is detailed in Chapter 6, "The Method Conclusion In Action."

## Initiating the method control virtual daemons

The method control virtual daemons will be initiated when the event correlation method is fired, i.e., the method firing condition is passed. However, to operate successfully there is an additional requirement; a location in the network topology model (specified by using the control location attribute) must be specified. Additionally, a containment level in the Containment Model can be specified by using the run in container attribute. This establishes the entities the method control virtual daemons will run on.

The operation of the method control virtual daemons is detailed in Chapter 6, "The Method Conclusion In Action."

## A final word on the method conclusion components

Before embarking on Chapter 6, "The Method Conclusion In Action," and discussing the method conclusion in action, the following summarizes how the two stages are initiated:

1. The actions stage is initiated by the construction of the action triggers which are all events on the entities identified by applying the control location to the network topology model and specifying a containment level.
2. The method control virtual daemons are run on all entities identified by applying the control location to the network topology model and specifying a containment level.

# Conclusion

This chapter has introduced the fundamentals behind event correlation and processing: event storage, filtering, correlation and action. These aspects form the foundation of the method template, which is the basis of all event correlation methods (the second of the AOC extensions) in Cisco Mobile Wireless Fault Mediator. This chapter has thoroughly explored the first three components of an event method: the method name, which is the basis of method chaining; the method firing condition, which determines when the method will be fired; and the method triggers, which determine how the final stage of an event correlation method, the method conclusion, is initiated.

The t components of a method conclusion—the actions stage and the method control virtual daemons—determine how the AMOS Events database is updated and whether external scripts can be launched as a result of the event correlation process. The following chapter examines all the attributes of the method conclusions and demonstrates how they can be applied.

CHAPTER 6

# The Method Conclusion In Action

This chapter discusses the final stage of an event correlation method, the method conclusion. There are two constituent elements within the method conclusion: the actions and the method control commands. The actions enable database updates to occur and the method control commands enable method control virtual daemons to be launched.

## The Final Step in the Process

So far, we have investigated how MONITOR uses a series of Polling Agents to examine the status of devices in the network, with poll definitions providing the Polling Agents with information about the strategies they should employ to achieve this task. As a result of polling, events are produced based on the attributes specified in the poll definitions poll list. MONITOR then broadcasts these events to AMOS for further processing; AMOS correlates these events with the network topology model and Containment Model and initiates a series of database updates depending on how the final stage of an event correlation method (the method conclusion) is constructed.

The method conclusion was identified in the last chapter, as were the triggers to its two components; the actions stage and the method control virtual daemons. This chapter explores these components, and how they operate, in more detail.

## The Actions Stage and its Structure

Processing within the actions stage determines how the event records in the AMOS Events database are manipulated. For instance, do we need to create a new event or alert based on the event correlation undertaken? Or do we need to alter some of the existing event or alert information? Or can we simply delete some of the existing records from the AMOS Events database?

The structure of the actions stage follows a similar pattern to an event correlation method. The components are as follows:

1. The actions triggers.
2. The action firing condition.
3. The actions.

# The Actions Triggers

Aspects of the actions triggers' construction were identified in the last chapter. To reiterate, the actions triggers are all events and alerts that have transpired on all devices (and entities contained within the devices) as a result of applying the network topology model and Containment Model. They are the input to the actions stage of an event correlation method.

# The Action Firing Condition

Similar to the method firing condition, the action firing condition is the final logical test to determine whether a specified action is executed. The action firing condition is applied to the actions triggers. Any event that satisfies the action firing condition gets passed to the respective action and gets updated based on the details specified. After the last of the actions has been run, the modified event is then injected into the AMOS Events database.

You can filter on any event record column names in the AMOS Events database using a series of logical operators (the operation of the action firing condition event filter being identical to the event filter in the method firing condition). A demonstration of the action firing condition is given in the practical example in the "Demonstrating Actions" section on page 6-7, which describes the application and operation of each action.

# The Actions

The actions allow you to manipulate the event records in the AMOS Events database. These actions are the foundation of event suppression. Cisco Mobile Wireless Fault Mediator contains three types of action:

- Create
- Change
- Delete

## Action Precedence

An event correlation method can contain multiple actions. For instance, an event correlation method may get triggered by a pingFail event; as a result, an alert is created and the original triggering event deleted. This type of action would involve using the create and delete actions, which raises the question of which action takes precedence.

Cisco Mobile Wireless Fault Mediator processes actions in the order they appear above (create, change, delete); after the delete action the AMOS Events Database is updated, with the created or modified events.

Action precedence therefore means that, while you can create an event and then delete it, you cannot do the reverse (delete, then create an event, for example). You must use method chaining to link two separate event correlation methods. A detailed demonstration of action precedence is presented later in the chapter after the following description of each action.

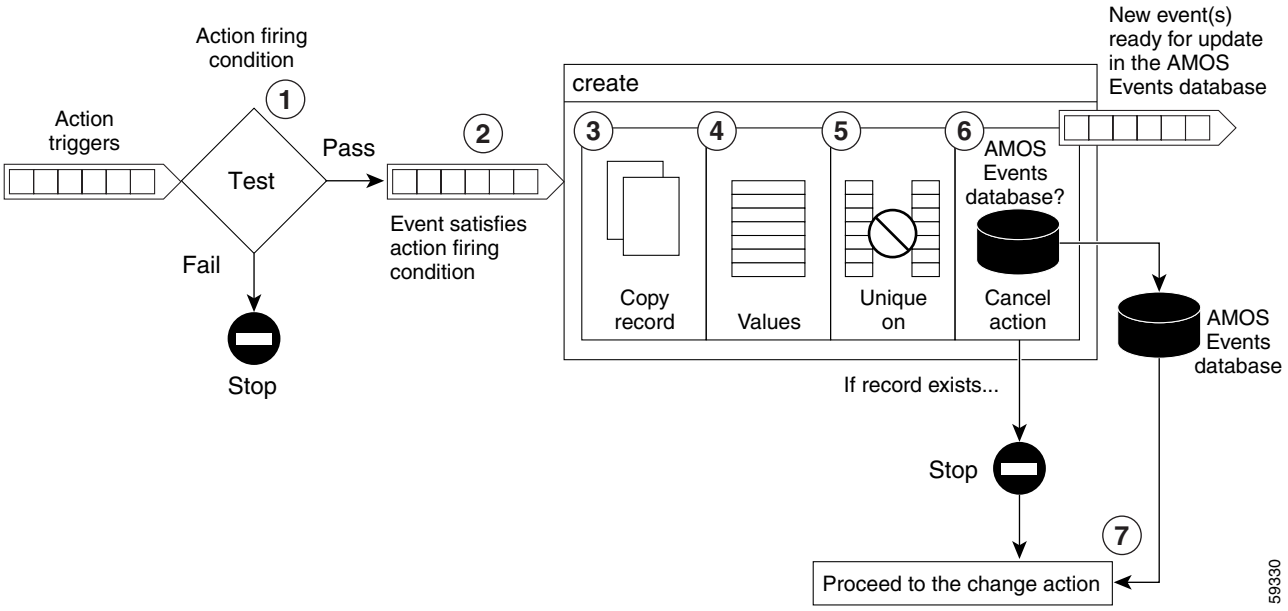
## Create

The create action allows you to create new events and inject them into the AMOS Events database. This type of action can increase the significance of an event or alert, or consolidate a series of events into a single new alert. Four attributes are available within the create action, described in Table 6-1.

***Table 6-1 The attributes available in the create action***

| Attribute | Description |
|---|---|
| Copy record | Specifies which event record is used as the basis for update. Three options are available:<br>• The method triggering event.<br>• The action triggering event.<br>• No event.<br>If the copy record attribute is set to either the method triggering event or action triggering event then all column names associated with either of these events are inherited by the created event unless overridden by the Values attribute (below).<br>Where no event is specified, the column names of the event created will be determined by the column names specified in the Values attribute. When specifying this option, you must ensure that values have been assigned to the column names of the event record which have a "not null" requirement. These are defined in Appendix A, "AMOS Databases." |
| Values | Specifies the column names of the copy record to amend. When using the values attribute, you must ensure the column names have new values assigned to them. Any column name and value specified in the values list will overwrite the value existing in the copy record. If you do not specify a column name and assign a value to it in the values column name, the new record will inherit the details from the copy record, i.e., they will remain unchanged. |
| Unique on | Specifies a set of column names that the new event must be unique on in order to appear as a "new" event in the AMOS Events database. If an event record already exists in the AMOS Events database, the Occurred and ChangeTime column names of the existing event are updated instead. |
| Cancel action | Allows you to construct a logical argument (using any column name of the AMOS Events database) to ensure that the event record you are about to create does not already exist in the AMOS Events database. If a match is found, the create action will not execute and the event correlation method will terminate. If there is no match, then the create action will execute. This attribute is particularly useful when using method chaining. |

*Figure 6-1 Attributes of the create action and their relationships*




The following describes how the attributes illustrated above interact, and presents a practical example of the create action.

1. The action firing condition is applied to the actions triggers.
2. An event exits the action firing condition and enters the create action.
3. The copy record attribute specifies which record is going to be the basis for the creation of a new event record; is it the method triggering event, the conclusions triggering event, or is a new event going to be constructed?
4. The values attribute defines the event record column names that are going to be overwritten on the copy record (remember, a full listing of the event record column names is available in Appendix A, “AMOS Databases”).
5. Unique on tests to determine whether the new event record exists already. If so, the Occurred and ChangeTime column names are updated. Otherwise, the new record is created.
6. Cancel action tests to determine whether an event record matching the newly created event record exists in the AMOS Events database; if so, then the event correlation method is aborted. Otherwise, the new event is cached ready for injection into the AMOS Events database after the other two actions have been processed.
7. Processing proceeds to the change action.

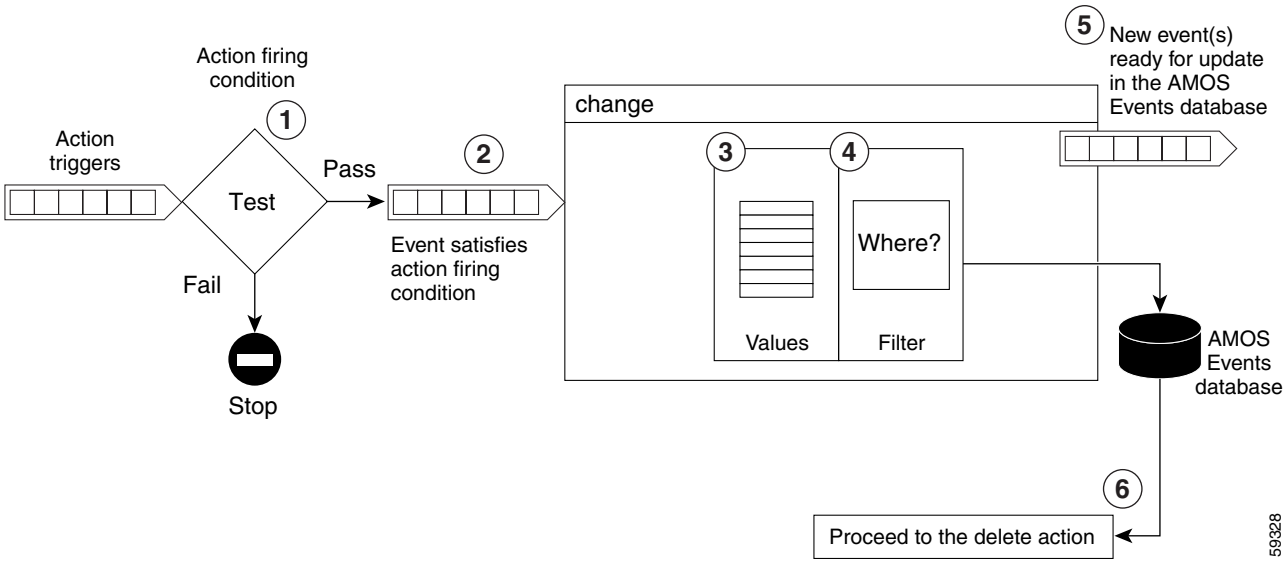
## Change

The change action enables parameters within an event record to be manipulated and updated without creating a new record; this type of action tends to be most useful in event suppression or escalation. Two attributes are available in the change action; these are described in Table 6-2.

*Table 6-2 The attributes available in the change action*

| Attribute | Description |
|---|---|
| Values | Specifies the column names of any event record to amend. When using the values attribute, you must ensure the column names have new values assigned to them. Any column name specified in the values list will overwrite the existing value. It must be used with the second attribute, filter. |
| Filter | In the same way as the **WHERE** clause in OQL, the filter attribute allows you to conditionally update event records in the AMOS Events database based on the column names and the values specified in the ***values*** attribute and a series of logical operators. |

*Figure 6-2 Attributes of the change action and their relationship*




Like the create action, the change action uses the actions triggers as its input. The sequence is as follows:

1. The action firing condition is applied to the actions triggers.
2. An event exits the action firing condition and enters the change action.
3. The values attribute is used to define the event record column names that are to be changed in those events that pass the action firing condition.

4. The filter attribute determines which events in the AMOS Events database are candidates for updates based on event record column names. The filter operates in the same way as the OQL WHERE clause (refer to the *Cisco Mobile Wireless Fault Mediator 2.1 - Topology and Platform Modeling Reference Guid*e for details).
5. After the filter is evaluated, the modified event is cached ready for update in the AMOS Events database after the delete action has been processed.
6. Processing proceeds to the delete action.

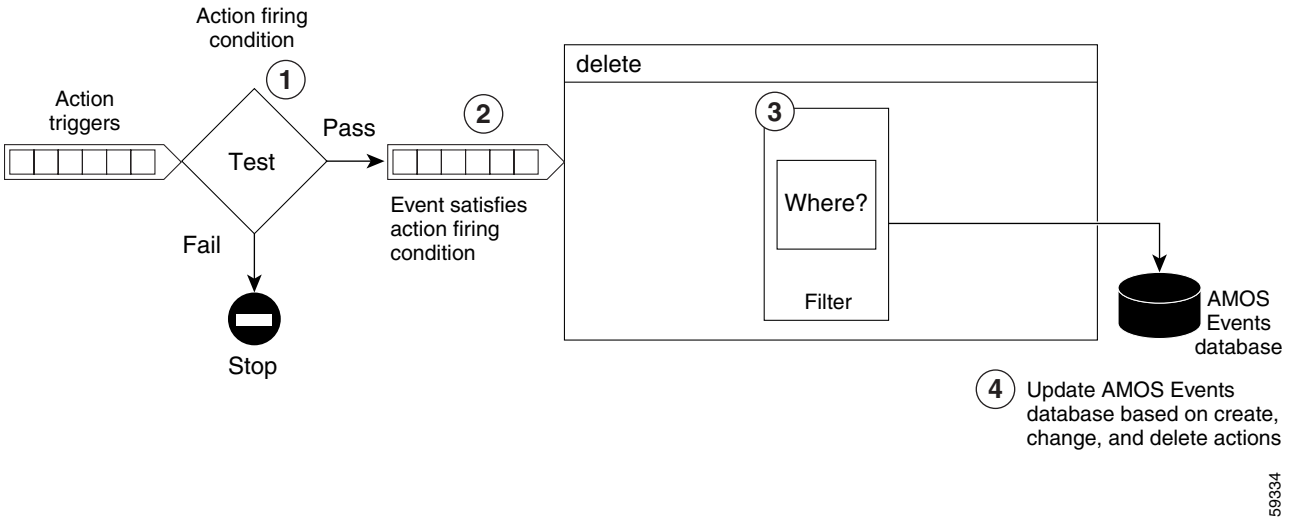
## Delete

The delete action enables event records in the AMOS Events database to be deleted. The operation of the delete action is the simplest of all the event modifiers; it only has a filter attribute.

***Table 6-3 The Attribute Available in the Delete Action***

| Attribute | Description |
|---|---|
| Filter | The filter functions in the same way as a **WHERE** clause in OQL. It allows you to delete event records in the AMOS Events database based on a conditional test using a series of logical operators. |

***Figure 6-3 Attributes of a delete method***




The delete action uses the actions triggers as its input; the sequence is then as follows:

1. The action firing condition is applied to the actions triggers.
2. An event exits the action firing condition and enters the delete action.
3. The filter attribute determines which events in the AMOS Events database are candidates for deletion.
4. Once the filter has been evaluated the AMOS Events database is updated based on the create, change and delete actions.

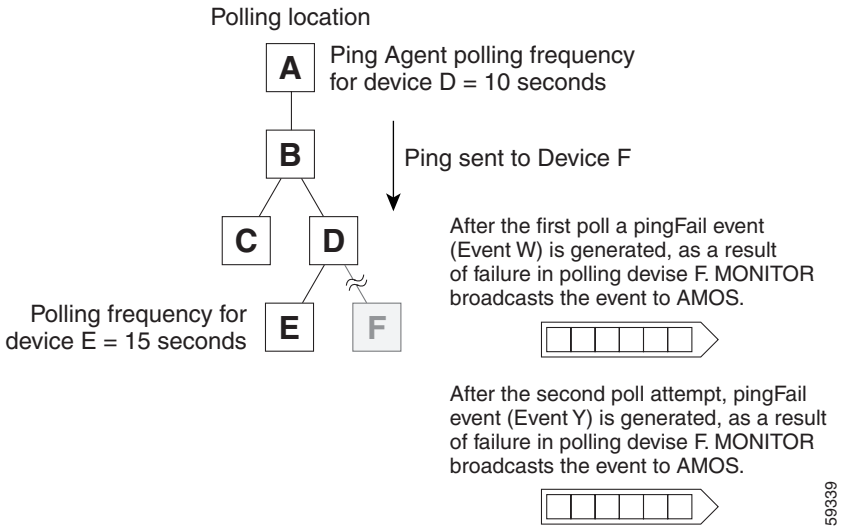
# Demonstrating Actions

In order to provide a clearer picture of the concepts behind the event correlation methods and method conclusion processing, the following practical example appears. In the example, we will create a realistic network scenario, defining an event correlation method, and step through the process flow identifying the impact of each attribute of an event correlation method.

## The Scenario

Consider the network shown in Figure 6-4:

***Figure 6-4 A typical network scenario***




In this network, the following facts can be seen:

- The AMOS Events database is initially empty.
- The polling location is Device A.
- The polling frequency is five minutes.
- Device F is a critical central server that has been disconnected from the network.
- Device F is polled twice by the Ping Polling Agent; this action generates two pingFail events, Event W and Event Y, associated with Device F.

# The Desired Outcome

Since Device F is so critical, the network administrator has decided that any pingFail event (indicating a connectivity concern) generated as a result of polling requires the following action:

1. A new event, an alert (Event X), should be created and injected in the AMOS Events database indicating Device A has failed a ping poll.
2. The first raw pingFail event (Event W) should be deleted from the AMOS Events database.
3. When a second pingFail event (Event Y) is received, the Occurred column name of the alert (Event X) created after the first pingFail should be changed (incremented by one) and the ChangeTime column name be updated.
4. The second raw pingFail event (Event Y) should be deleted from the AMOS Events database.

# The Event Correlation Method—"Create"

To implement the chain of actions described above, the network administrator begins by defining the following event correlation method in Device F's AOC file (although the event correlation method could be defined higher in the AOC class hierarchy and inherited by Device F too). The method used in this example is arbitrarily called "Create" and is inherited from the "Core" class at the top of the AOC hierarchy. The method has the following attributes:

- It is a non-timed method.
- Method chaining is not used.
- The firing policy is empty, i.e., it always fires.
- The control location is set to "instance".
- The Containment Model is not utilized.
- The method firing condition and respective conclusion actions (create, change and delete) within the event correlation method are defined as follows:

## The Method Firing Condition

*Table 6-4 The event filter of the method firing condition*

| Attribute | Value |
|---|---|
| Event filter | `Severity > eval(int,'$UNKNOWN') AND`<br>`InternalAction = eval(int,'$NONE') AND`<br>`EventType = eval(int,'$EVENT') AND`<br>`EventName = 'pingFail' AND`<br>`ActionType = eval(int,'$NEW')` |

## The Create Action

*Table 6-5 The values assigned to the create action attributes*

| Attribute | Value |
|---|---|
| Conclusion firing condition | `EventId = eval(int,'&&EventId')` |
| Copy record | `0` |
| Values | `EventType = eval(int,'$ALERT'),`<br>`CreateTime = eval(long,'$TIME'),`<br>`CauseType = eval(int,'$SYMPTOM'),`<br>`Occurred = 1` |
| Unique on | - |
| Cancel action | `EntityName = eval(text,'&&EntityName')AND`<br>`EventName = eval(text,'&&EventName') AND`<br>`EventType = eval(int,'$ALERT')` |

## The Change Action

*Table 6-6 The values assigned to the change action attributes*

| Attribute | Value |
|---|---|
| Conclusion firing condition | `EventName = eval(text,'&&EventName') AND`<br>`EventType = eval(int,'$ALERT')` |
| Values | `ChangeTime = eval(long,'$TIME'),`<br>`Occurred = eval(int,'(&Occurred) + 1'),` |
| Filter | `EntityName = eval(text,'&EntityName') AND`<br>`EventName = eval(text,'&&EventName') AND`<br>`EventType = eval(int,'$ALERT')` |

## The Delete Action

*Table 6-7 The Values Assigned to the Delete Action Attributes*

| Attribute | Value |
|---|---|
| Conclusion firing condition | `EventId = eval(int,'&&EventId')` |
| Filter | `EventId = eval(int,'&&EventId')` |

# The Process Flow

At the start of the process flow, the Polling Agents of MONITOR determine that Device F is unreachable and generate an event based on the details specified in the event construction list of the poll definition. In this case the event construction list will specify a pingFail event, Event W.

## The first pingFail event—Event W

The relevant column names and values of Event W can be seen in Table 6-8 below.

*Table 6-8 Event W—the relevant column names and values*

| Column name | Value |
|---|---|
| EventId | 1 |
| EntityName | Device F |
| EventType | Event |
| EventName | pingFail |
| Severity | Major |
| CreateTime | 05:30 16 MAR 2001 |
| ChangeTime | 05:30 16 MAR 2001 |
| ActionType | New |
| InternalAction | None |
| CauseType | Symptom |
| Occurred | 1 |

1. MONITOR sends Event W to AMOS. On receipt, Event W is injected into the AMOS Events database.
2. The event's injection into the AMOS Events database triggers the event correlation method "Create"; defined in the AOC associated with Device F.
3. The "Create" method firing condition is applied to Event W to determine whether the remainder of the method should proceed. In "Create", the conditional test consists of an event filter only. Event W satisfies this criteria since it is a new pingFail event with a Severity of major. Processing then moves to the next stage, construction of the actions triggers, by applying the control location.
4. Since the control location is "instance", the actions triggers will be all events in the AMOS Events database associated with the instance of the device that generated Event W, i.e., Device F. In this case, however, there is only Event W (the same event that triggered the event correlation method). Processing then continues to the actions themselves— create, change, and delete.

5. The action firing condition for the create action tests to see if the EventId of the event in the AMOS Events database is equal to the EventId of the method triggering event (denoted by the use of &&EventId in the eval statement). In this example it does, since they are one and the same (Event W).
6. The copy record attribute is set to “0”, which means that the method triggering event (Event W) will be used as the basis for creating the new event record.
7. Four column names in the values attribute (EventType, CreateTime, CauseType, and Occurred) will be overwritten with the values evaluated from the eval statements in the respective column names. In addition, the ChangeTime will be assigned the same value as the CreateTime and the EventID will be the next sequential ID number available in the system. The newly created event, Event X, will appear as show in Table 6-9.

***Table 6-9 Event X—the relevant column names and values***

| Column name | Value |
|---|---|
| EventId | 2 |
| EntityName | Device F |
| EventType | Alert |
| EventName | pingFail |
| Severity | Major |
| CreateTime | 05:31 16 MAR 2001 |
| ChangeTime | 05:31 16 MAR 2001 |
| ActionType | New |
| InternalAction | None |
| CauseType | Symptom |
| Occurred | 1 |

8. Processing then continues to the unique on attribute. In this case, unique on is empty and processing will move straight to the cancel action attribute.
9. The cancel action attribute checks the AMOS Events database to see if an event record exists which matches Event X. In this case, the AMOS Events database contains only Event W, which is different from Event X; processing of the create action will stop and Event X is cached ready for insertion into the AMOS Events database after the other two actions (change and delete) have been processed.
10. The actions trigger for the change action is Event W, the same event that triggered the create action.
11. The action firing condition for the change action specifies that the EventName must be the same as the EventName of the method triggering event and it must be an alert. At this stage, there are no events in the AMOS Events database that match that criteria (since Event X has not been injected yet), so processing moves to the delete action.

12. Similar to the create and change actions, the actions trigger for the delete action is Event W.
13. The action firing condition for the delete action is the same as the one for the create action, i.e., the EventId must equal the EventId of the event that triggered the method (specified by the use of &&EventId in the eval statement).
14. The filter is then applied to all the events in the AMOS Events database. In this example, it looks for any event in the database with the same EventId as the method triggering event, Event W (specified by the use of &&EventId in the eval statement) and prepares to delete it. In this case, the event to be deleted will be Event W.
15. The final step in the event correlation method is to update the AMOS Events database. In this case, Event X is injected into the AMOS Events database and Event W is deleted. The AMOS Events database now contains just Event X.

## The Second PingFail Event—Event Y

Five minutes after the first poll, Device F is polled again. Since the device is still down, a second pingFail identical to Event W (but referred to as Event Y) is generated.

The relevant column names and values of Event Y can be seen in Table 6-10 below.

*Table 6-10 Event Y—the relevant column names and values*

| Column name | Value |
|---|---|
| EventId | 3 |
| EntityName | Device F |
| EventType | Event |
| EventName | pingFail |
| Severity | Major |
| CreateTime | 05:35 16 MAR 2001 |
| ChangeTime | 05:35 16 MAR 2001 |
| ActionType | New |
| InternalAction | None |
| CauseType | Symptom |
| Occurred | 1 |

1. Again, MONITOR sends the event to AMOS via the event stream. As before, Event Y is injected into the AMOS Events database and the event correlation method “Create” associated with Device F is rerun, this time with Event Y as the method triggering event. The AMOS Events database now contains Event X and Event Y.
2. Again, the method firing condition is satisfied and the actions triggers constructed. Because the control location is set to “instance”, the actions triggers will be all events in the AMOS Events database associated with the instance of the device (Device F) that generated Event Y. In this case there will be two events: Event Y (the new method triggering event) and Event X (the pingFail alert). Processing of the three actions then commences with Event Y and Event X as inputs.

3. The action firing condition for the create action specifies that the EventId must equal the EventId of the event that triggered the method (denoted by the use of &&EventId in the eval statement). The only event that satisfies this criteria is Event Y.
4. The copy record attribute is set to "0", which means the method triggering event (Event Y) will be the event record used as the basis for creating the new event record. The four column names in the values attribute (EventType, CreateTime, CauseType, and Occurred) will be overwritten with the values evaluated from the eval statements in the respective column names. The new event, Event Z, will be a pingFail alert and designated as a symptom. Event Z will be cached ready for insertion into the AMOS Events database with the following details:

*Table 6-11 Event Z—the relevant column names and values*

| Column name | Value |
|---|---|
| EventId | 4 |
| EntityName | Device F |
| EventType | Alert |
| EventName | pingFail |
| Severity | Major |
| CreateTime | 05:36 16 MAR 2001 |
| ChangeTime | 05:36 16 MAR 2001 |
| ActionType | New |
| InternalAction | None |
| CauseType | Symptom |
| Occurred | 1 |

5. Processing will move to the cancel action attribute. The cancel action attribute will check the AMOS Events database to see if an event record exists that matches Event Z. On this occasion, a match is found since the pingFail alert, Event X (created when the first pingFail event occurred) is identical to Event Z; further processing of the create action is terminated and Event Z is removed from the cache to the AMOS Events database.
6. Processing of the change action then commences. As with the create action, Event X and Event Y will be the inputs to the change action.
7. The action firing condition for the change action specifies the EventName must be the same as the EventName of the event that triggered the event correlation method, and it must be an alert. There is only one event in the AMOS Events database that satisfies this criteria, Event X.

8. The values attribute is then processed. This changes the ChangeTime column name of Event X to the time in the CreateTime column name of Event Y and increments the Occurred column name by one on any events in the AMOS Events database that satisfy the filter, which seeks those events where the EntityName is the same as the event record that triggered the change action or an event with the same name as the method triggering event (denoted by the use of &&EventName). The only event in the AMOS Events database that satisfies these criteria is Event X. Event X is cached (ready for update after processing of the delete action) with updated ChangeTime and Occurred column names. This update indicates there have been two pingFails of an alert status associated with Device F, and the last one transpired at the time inserted into the ChangeTime column name of the event record. Event X will appear as follows:

*Table 6-12 Event X—the relevant column names and values*

| Column name | Value |
|---|---|
| EventId | 2 |
| EntityName | Device F |
| EventType | Alert |
| EventName | pingFail |
| Severity | Unknown |
| CreateTime | 05:31 16 MAR 2001 |
| ChangeTime | 05:36 16 MAR 2001 |
| ActionType | New |
| InternalAction | None |
| CauseType | Symptom |
| Occurred | 2 |

9. Processing of the delete action then commences. The actions trigger for the delete action is Event X. The action firing condition for the delete action is the same as the one for the create action, i.e., the EventId must equal the EventId of the event that triggered the method (denoted by the use of &&EventId in the eval statement), which is the case.
10. As before, the filter is applied to all events in the AMOS Events database. In this instance, it looks for any event in the database with the same EventId as the method-triggering event and prepares it for deletion. In this case the event to be deleted is Event Y, the method-triggering event itself.
11. The final step is to update the AMOS Events database with the actions performed above. Hence, Event X (the pingFail alert) will be updated with the values identified in the values column name of the change action and Event Y (the second pingFail event) will be deleted. At the end of this process there will be one event record in the AMOS Events database, Event X, with an Occurred column name equal to two and a ChangeTime column name equal to the CreateTime of Event Y.

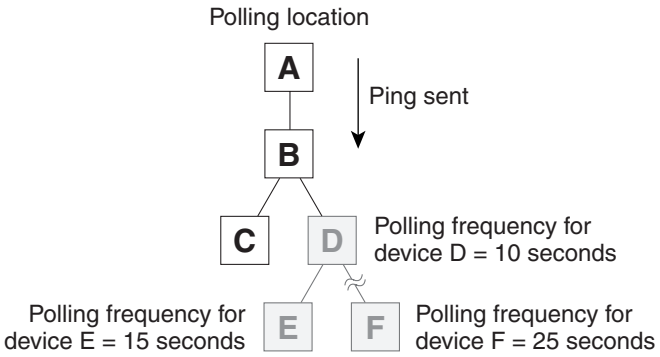
# The Method Control Command—Running Virtual Daemons

The method control command allows you to associate virtual daemons with devices on the network. In the context of an event correlation method, a virtual daemon is a process which enables the actions stage of the method conclusion to run on devices specified by the control location without having to re-run the entire event correlation method. In essence, the process is a daemon; however, it is considered virtual since it does not actually run on the device itself—rather, it runs in association with the instance of the object in the network topology model. This section establishes why virtual daemons are required, their application and how they work by presenting a scenario and then working through the process flow.

## The Scenario

Consider the network shown in Figure 6-5:

***Figure 6-5 The example network***




Timeline

| Time(secs) | Device | Event | Alert | Comment | Steps |
|---|---|---|---|---|---|
| 0 | D | ~~P~~ | Q | Occurred field of Q equals 1 | 1 – 2 |
| 5 | – | – | – | – | |
| 10 | D | ~~R~~ | Q | Occurred field of Q equals 2 | 3 |
| 15 | F | ~~S~~ | T | – | 4 |
| 20 | D | ~~U~~ | Q | Occurred field of Q equals 3 | 5 – 13 |
| 25 | F | ~~V~~ | W | Escalate triggered; Q becomes root-cause alert; SuppressDown Stream triggered; Event T suppressed. | 14 – 21 |

59335

In this illustration you can see that:

- The polling location is Device A
- The polling frequency for Device D is 10 seconds
- The polling frequency for Device E is 15 seconds
- The polling frequency for Device F is 25 seconds
- Devices D, E and F are all instances of an AOC that uses four event correlation methods:
  - Create—is inherited from the parent class Core. It creates a new event, an alert, when a ping fail event is received from a device managed by its policies and deletes the method-triggering ping fail event.
  - Escalate—is inherited from the parent class Core. It escalates the Severity of the ping fail alert generated by Create to "root cause" if a device managed by its policies generates more than two ping fail events of alert status (i.e., when the Occurred column name is 3).
  - SuppressDownStream—is inherited from the parent class Core. It suppresses alerts on any subservient devices. In addition, SuppressDownStream contains a method control command that specifies a virtual daemon to run.
  - Restore—is inherited from the parent class Core. It deletes ping fail alerts on any subservient devices from the AMOS Events database when it receives a clear event. In addition, when the Restore method is triggered it also terminates any virtual daemons running.
- Method chaining is employed; Escalate is evaluated when Create is fired, and SuppressDownStream is evaluated when Escalate is fired.
- Device D has been disconnected from the network

# How the scenario evolves

1. Device D is polled when required by the AOC that manages Device D. Since Device D has been disconnected from the network, a ping fail event, Event P, is generated by MONITOR. The content of Event P will depend on the details contained in the event construction list of the poll definition. Event P is then sent to AMOS.
2. When Event P is injected into the AMOS Events database it triggers the event correlation method Create, which creates a new event, Event Q, a ping fail alert that indicates Device D is unreachable. Event Q is inserted into the AMOS Events database and Event P is deleted.
3. After an elapsed time of 10 seconds, Device D is re-polled. Again it returns a ping fail event, Event R, that MONITOR sends to AMOS. Event R is inserted into the AMOS Events database and Create triggered. On this occasion a matching alert exists, Event Q. The change action is used to increment the Occurred column name of Event Q by one to two, and the triggering event, Event R, is deleted from the AMOS Events database. The content of the AMOS Events database remain just Event Q, an alert on Device D.
4. After an elapsed time of 15 seconds, Device E is polled. Since Device E is isolated as a result of loss of connectivity to Device D, a ping fail event, Event S, will be generated by MONITOR and sent to AMOS. Injecting Event S into the AMOS Events database triggers Create and a new event, Event T (a pingFail alert), is created and inserted into the AMOS Events database. Event S is deleted. The contents of the AMOS Events database will be Event Q, an alert on Device D, and Event T, an alert on Device E.

5. After 20 seconds, Device D is re-polled. As before, it returns a ping fail event, Event U, that MONITOR sends to AMOS. As in step 3., the insertion of Event U into the AMOS Events database triggers Create. Event Q still exists, so its Occurred column name is incremented by one to three and the triggering event, Event U, is deleted from the AMOS Events database. The contents of the AMOS Events database will be Event Q, an alert on Device D, and Event T, an alert on Device E.
6. Since the Occurred column name of Event Q (the pingFail alert) is now three and Create and Escalate are chained, Escalate is triggered; Escalate uses the change action to escalate the Severity of Event Q to critical, and the CauseType to root. The contents of the AMOS Events database will be Event Q, a root cause alert on Device D, and Event T, an alert on Device E.
7. Since Escalate and SuppressDownStream are chained, SuppressDownStream is triggered when processing of Escalate is complete.
8. The SuppressDownStream event correlation method uses the change action to downgrade the Severity of pingFail alerts on devices subservient to Device D. In this case one event will have its column names changed, the pingFail alert Event T, on Device E. The AMOS Events database will thus contain Event Q, a root cause alert on Device D, and ***Event T***, a suppressed alert on Device E.
9. After an elapsed time of 25 seconds, Device F will be polled. Since Device F is isolated as a result of loss of connectivity to Device D, a ping fail event, Event V, will be generated by MONITOR and sent to AMOS. As a result of Event V being injected into the AMOS Events database, Create is triggered and a new event, Event W (a pingFail alert), is created and inserted into the AMOS Events database. Event V (the triggering event) is deleted.
10. The contents of the AMOS Events database will be Event Q, a root cause alert on Device D; Event T, a suppressed alert on Device E; and Event W, an alert on Device F.

In the example above, a virtual daemon can be configured in the SuppressDownStream event correlation method. With the virtual daemon running in association with devices subservient to the device that generated the original event, Device D, any event transpiring one of these subservient devices will trigger the re-running of the conclusions of SuppressDownStream (which suppresses alerts).

Therefore, when Device F is polled after an elapsed time of 25 seconds and Event V generated as a consequence, the new sequence of operation will be as follows:

1. When Event V is injected into the AMOS Events database, it triggers Create, which creates a new event, Event W, a ping fail alert that indicates Device F is unreachable. Event W is injected into the AMOS Events database and Event V is deleted. The AMOS Events database contains Event Q, a root cause alert on Device D; Event T, a suppressed alert on Device E; and Event W, an alert on Device F.
2. When Event W is injected into the AMOS Events database, it will trigger the SuppressDownStream conclusions to be re-run. Event correlation method processing stops and the virtual daemons are placed on standby.
3. The contents of the AMOS Events database will be: Event Q, a root cause alert on Device D; Event T, a suppressed alert on Device E; and Event W, a suppressed alert on Device F.
4. The virtual daemons will continue to run until another event correlation method intervenes indicating connectivity to Device D has been restored. In this case, the event correlation method that performs this task is Restore. The Restore method is triggered by a clear event; once triggered, an instruction is sent to terminate all virtual daemons running in association with subservient devices.

The advantage of using virtual daemons is that they are asynchronous and are only activated when an event transpires; in this case, the pingFail alert (Event W) on Device F. This aspect means processing efficiency is not compromised and sequencing issues are avoided. Additionally, because of the class-based nature of the system it is possible to define a method at the top of the AOC hierarchy to handle such situations, much like Create, Escalate, SuppressDownStream and Restore in the above example.

Table 6-13 shows the three attributes used to control the virtual daemons used in the method control command.

***Table 6-13 Method control command attributes***

| Attribute | Description |
|---|---|
| Daemon running order | Determines whether the virtual daemons are considered before or after the event correlation methods on a particular device. In the example above, the virtual daemons were run after the SuppressDownStream event correlation method had completed its conclusion actions. |
| Set daemon | Logical flag to determine whether the virtual daemon is activated or not; true denotes the virtual daemon is activated, false denotes deactivated. |
| Clear daemon | The only valid entries in this field are other valid method names, for instance, “methodName”. When methodName is specified in the clear daemon field the virtual daemon will terminate when methodName is fired on the device the virtual daemon is running on. |

# Method Conclusion Precedence

The ability of Cisco Mobile Wireless Fault Mediator to construct methods that perform more than one task can often prove essential to a busy network administrator. For instance, it is possible to create an event record and delete others, or change column names of an event record and run a script to launch an email application, all in the same event correlation method. Cisco Mobile Wireless Fault Mediator executes components of the method conclusion in the following order:

1. The actions;
    d. Create
    e. Change
    f. Delete
2. The method control command.

This is identical to action precedence; you can utilize the actions and then run a directive in the same method.

# Conclusion

This chapter has provided an insight into the operation of the final stage of the event correlation methods—the method conclusion, its two components of an event correlation method conclusion, and their attributes.

The actions stage manipulates and updates events in the AMOS Events database by utilizing three actions (create, change and delete) and the attributes and operation of the method control commands allow virtual daemons to be launched that prevent subservient devices from generating a series of downstream alerts when an upstream device has failed and the event correlation method on that device has already been fired.

The next chapter examines how the events generated as a result of the polling process, and the alerts produced by the event correlation process, are displayed to the user.

CHAPTER 7

# Robust Trap Transport Mechanism

MWFM now supports the Robust Trap Transport mechanism between MWFM and the NMS. Every MWFM Trap includes a MIB variable viz rivSequentialID which contains the MWFM Trap sequence number.

For understanding purpose, here is sample MWFM trap sent out of MWFM

```
received trap from [172.19.26.14].39173
    community: public
   enterprise: 1.3.6.1.4.1.2517
   agent addr: 172.19.26.14
   generic ID: 6
  specific ID: 2
       uptime: 116 days, 23:14:40
     bindings: rivSequentialID           => 1
               rivEntityName             => 12.12.12.7[ 0 [ 0] ].WIRELESS
               rivCreateTime             => 1010603807
               rivDescription            => Ping fail on device 12.12.12.7[ 0 [0] ]
               rivCauseType              => 2 (root)
               rivEventId                => 8
               rivLocation               =>
               rivSeverity               => 5 (critical)
               rivExtraInfo              =>
               rivClassName              => Interface
               rivEventName              => pingFail
               rivEventType              => 2 (alert)
               rivContact                =>
               rivAssignedTo             =>
               rivAcknowledged           => 0
               rivCorrelatedId           => []
               rivEventGroupId           => 0
               rivActionGlyph            => none
               rivOccurred               => 4
               rivFaultActionType        => 1 (update)
               rivAgentAddress           => mw-nms-u10
               rivInternalAction         => 0
```

Range of the rivSequential ID can be configured by the user. For example, if the user has configured the range as 1 to 100, then MWFM will send traps with rivSequentialID= 1 up to 100. And then it will roll over to 1.

The NMS can determine if it has lost some MWFM traps by looking at the sequence number of MWFM Trap. Under no loss conditions, all the MWFM traps will be received sequentially one after the other i.e. rivSequenceID=1,2,3...10. If the NMS looses some MWFM Traps, then it can request MWFM to re-send the traps using the Java API.


# AMOS Databases

This appendix provides a complete listing and description of the column names of the AMOS databases, the AMOS Events database, the AMOS Entity database, and the AMOS Class database.

# AMOS databases

To enable Cisco Mobile Wireless Fault Mediator to achieve its objective—event correlation with the network topology model and Containment Model to determine root cause—and function efficiently, AMOS uses a series of databases to store information received from other MWFM NMOS components. Each of these databases contains a series of column names which are the basis for event correlation. There are three main databases:

1. The AMOS Events database, which contains event information.
2. The AMOS Entity database, which contains entity information.
3. The AMOS Class database, which contains class information.

## The AMOS Events database (mojo.events)

The AMOS Events database contains a log of all information pertaining to event records generated by MONITOR. When AMOS is launched from the command prompt, event information is read from STORE into the AMOS Events database. Within each record, there are a series of column names which are used in many of the conditional filters when constructing an event correlation method; Table A-1 below describes them all.

***Table A-1 Column names available in the AMOS Events database***

| Column name | Data type | Column constraint | Description |
|---|---|---|---|
| EventId | Long Integer | PRIMARY KEY NOT NULL | The event ID |
| EntityName | Text | PRIMARY KEY NOT NULL | The name of the associated Active Object. |
| ClassName | Text | PRIMARY KEY NOT NULL | The name of the associated Active Object Class. |
| Description | Text | | A textual description of the event. |
| EventName | Varchar (256) | | The name of the event. |

*Table A-1 Column names available in the AMOS Events database (continued)*

| Column name | Data type | Column constraint | Description |
|---|---|---|---|
| EventType | Integer | Externally-defined eventType data type | Type of event: $EVENT (0), $DATA (1) or $ALERT (2). |
| Severity | Integer | PRIMARY KEY NOT NULL Externally-defined severity data type | The OSI severity code. $CLEAR (0), $UNKNOWN (1), $WARNING (2), $MINOR (3), $MAJOR (4), $CRITICAL (5), $NONE (6). |
| Contact | Text | | The contact group responsible for the device that generated the event. |
| AssignedTo | Text | PRIMARY KEY NOT NULL | The person the event has been assigned to. |
| Acknowledged | Integer of boolean type | PRIMARY KEY NOT NULL | Denotes whether the event has been unacknowledged or acknowledged. $UNAKNOWLEDGED (0), $ACKNOWLEDGED (1). |
| Location | Text | | Location of the device that generated the event. |
| CorrelatedId | List of Long Integers | | List of associated event ID's. |
| EventGroupId | Long Integer | PRIMARY KEY NOT NULL | List of associated events. |
| ActionGlyph | Text | | An alert display glyph (icon). |
| CreateTime | Long Integer | | Time of the first occurrence of the event. |
| ChangeTime | Long Integer | | Time of the last occurrence of the event. |
| Occurred | Integer | | Number of identical events that have occurred. |
| CauseType | Integer | Externally-defined causeType data type | The type of alert. Can be $CAUSEUNKNOWN (0), $SYMPTOM (1) or $ROOT (2). |
| ActionType | Integer | Externally-defined actionType data type | The type of action the event represents a new event, updated event or deleted event. $NEW (0), $UPDATE (1), $DELETE (2). |
| InternalAction | Integer | Externally-defined internalAction data type | The internal action associated with the event $ACK (1), $UNACK (2), $ASSIGN (3), $TOOL (4), $INTCLEAR (5), $CLEARALL (6), $CLEARCHAIN (7). |
| AgentAddress | Text | | Polling Agent location (i.e., the IP address of the Polling Agent). |
| ExtraInfo | Object | Externally-defined varbind list | List of additional information. |

# The AMOS Entity database (topoCache.entityByName)

The AMOS Entity database contains all information pertaining to network entities, their containment and connections. When AMOS is launched, entity information is read from MODEL into the AMOS Entity database. This information is particularly useful when applying the conditional tests and filters in the poll definitions or event correlation methods, as it enables you to differentiate objects of the same class from one another.

***Table A-2 Column names available in the AMOS Entity database***

| Column name | Data type | Column constraint | Description |
|---|---|---|---|
| ObjectId | Long Integer | PRIMARY KEY NOT NULL | Unique Object ID of the network object. |
| EntityName | Text | PRIMARY KEY NOT NULL | Unique descriptive name of a network object. |
| Address | List of text data type | | List of OSI model Layer 1-7 addresses for the object. |
| Description | Text | | Value of sysDescr MIB variable or other suitable description of the object. |
| EntityType | Integer | Externally-defined entityType data type | Element type of the object. |
| ClassName | Text | | The associated Active Object Class (if applicable). |
| EntityOID | Text | | Value of sysOID MIB variable of the object. |
| ExtraInfo | Object | Externally-defined varbind list | List of additional information. |
| Status | Integer | Externally-defined status data type | Flag showing status of the network object. |
| Security | Text | | Password to access network entity (if applicable). |
| RelatedTo | List of Text data type | | List of connections to the network object. |
| Contains | List of Text data type | | List of elements or other containers contained by this object. |
| PartOfName | List of text data type | | The name of the physical container to which the network object belongs. |
| IsActive | Integer of boolean type | | Flag indicating whether an Active Object Class is needed. |
| CreateTime | Time | | Creation time of network object record in table. |
| ChangeTime | Time | | Time of last modification to the network object record. |
| ActionType | Integer | Externally-defined actionType data type | Type of record. |

# The AMOS Class database (class.activeClasses)

The AMOS Class database contains information pertaining to an instance's class—instantiation information, parent class information, data dictionary details, and the extensions. When AMOS is launched, class information is read from CLASS into the AMOS Class database, whose column names are listed in Table A-3.

***Table A-3 Column names available in the AMOS Class database***

| Column name | Data type | Column Constraint | Description |
|---|---|---|---|
| ClassName | Text | PRIMARY KEY NOT NULL | The name of the Active Object Class (AOC). |
| SuperClass | Text | NOT NULL | The name of the parent class of the class specified by the ClassName column name. |
| Dictionary | List of Text data type | | A list of data dictionaries used by the AOC. |
| Instantiate | Text | NOT NULL | The rules for instantiating the AOC. |
| Extensions | Object type extension | | List of the extensions associated with the AOC; these include the poll definitions and event correlation methods. |
| ActionType | Integer of type actions | Externally-defined actionType data type | The broadcast ID type. |


# The Eval Statement

This section describes the syntax and functionality of the ***eval*** statement, which is used in the AOC extensions. It is also used in the Object Query Language (OQL) and the Stitcher language of MWFM NMOS. The **eval** statement provides a very versatile way of accessing column names of database records, MWFM NMOS, and Cisco Mobile Wireless Fault Mediator.

## Introduction

This chapter introduces the syntax of the **eval** statement. The **eval** statement can be used in the AOC extensions of Cisco Mobile Wireless Fault Mediator (OQL and the Stitcher language in MWFM NMOS). Though you should have no problem in understanding the defined syntax and functionality, it is recommended that you take advantage of the information provided in the Language Prerequisites chapter of the MWFM NMOS Reference Guide, which defines various terms and the use of different quotation marks within the **eval** statement, before you proceed.

## Ampersands and the eval Statement

We have already seen several syntactical statements arise used to construct an event correlation method's method conclusion, such as eval statements or ampersands (&) for distinguishing scope.

Scope, in the context of the event correlation methods, refers to the triggers used to initiate the event correlation method and the method conclusion components. The method triggers can be thought of as the outer scope (since it is not possible to initiate the method conclusion without the method), and the actions triggers the inner scope.

**Note** Remember, the method triggers and actions triggers are simply events, or more precisely event records, that contain a series of column names. The use of ampersands (&) provides a means of referencing these column names in the respective triggers.

Only the actions stage and the directive use ampersands (and **eval** statements), and the meaning of an ampersand (&) depends on which method conclusion component is active. For instance, a single ampersand (&) used in a create action of the actions stage has a different meaning to a single ampersand (&) used in the directive.

# Ampersands and the Action Stage

To refer to an event record column name in the actions triggers from the actions stage, a single ampersand (&) symbol is used; to refer to an event record column name in the method triggers from the actions stage, a double ampersand (&&) is used instead. In this way, you can use the symbol to step in and out of the inner and outer scope as needed. As an example, consider the following condensed event records:

*Table B-1 Method triggering event*

| Column name | Value |
|---|---|
| EventId | 1 |
| EventType | Event |
| EventName | pingFail |
| Severity | Major |
| Occurred | 1 |

*Table B-2 Action triggering event*

| Column name | Value |
|---|---|
| EventId | 5 |
| EventType | Event |
| EventName | pingFail |
| Severity | Critical |
| Occurred | 3 |

The first of the following **eval** statements would evaluate to a value of 5, because it uses a single ampersand to access the action triggering event:

```
"EventId = eval (int, '&EventId')"
```

Using a double ampersand, however, accesses the method triggering event instead:

```
"EventId = eval (int, '&&EventId')"
```

This second statement thus evaluates to 1.

# The eval Statement

The **eval** statement provides a powerful way of accessing MWFM NMOS and Cisco Mobile Wireless Fault Mediator variables, as well as referencing the values of database records. The general format of the **eval** statement, followed by an example, is shown below:

```
eval ( <cast_operator> , <evaluation_clause> )
eval ( long , &objectId )
```

**eval** statements are very flexible, because they allow you to combine multiple expressions to form component parts of complex algebraic expressions. Those component parts are described below.

## The Cast Operator

The cast operator (represented in the example above by the data type 'long') forces the data type returned by the evaluation of the operator to convert the data type required. The cast operator is any legal OQL data type definition, e.g., **OBJECT TYPE, LIST TYPE, INTEGER, FLOAT, STRING, VARCHAR (n), CHAR, TEXT**, etc. Table B-4 on page B-4 contains the full list of data types.

## The evaluation clause

The evaluation clause is the expression that is to be evaluated and coerced into the data type specified by the cast operator. The evaluation clause supports various evaluation expressions, which may be combined algebraically to form more complex expressions. The evaluation expressions may be one or a combination of the following:

- A reference to a variable (identified as being preceded by a dollar sign ($)). The variables that can be referenced may be global variables, which include system built-in variables such as **$TIME;** NMOS-defined and Cisco Mobile Wireless Fault Mediator-defined variables, such as **$ObjectName;** and any environment variables of the shell within which the NMOS executable is running, e.g., **$SHELL**.
- A reference to specific column names of database records known to the current scope, are identified as being preceded by the ampersand symbol (&). Examples of references to database records within the current scope are **&EntityName, &ClassName, &EntityOID,** etc.
- A reference to specific column names of database records outside the current operational scope, identified by being preceded by multiple ampersand symbols (&) as described in the "Ampersands and the eval Statement" section on page B-1. An example of such a reference might be **&&EventId**.
- A manipulative statement, which can modify complex data types such as lists or objects; concatenate two items together, delete item(s) from a list etc.
- An extraction of an entry within a complex database record using the arrow operator.
- A special expression using the THIS keyword to move in and out of the Containment Model.
- Algebraic combinations of multiple evaluation expressions.

# Evaluation clause keywords

The following keywords are valid within **eval** statements:

*Table B-3 Table of valid eval statement keywords*

| Keyword | Synopsis of keyword function |
|---|---|
| APPEND | Allows you to append data to a list or object. |
| APPENDUNIQUE | Allows you to append data to a list or object only if it does not already exist within the list. |
| CAT | Allows you to create new data by concatenating a series of items together. |
| DELETE | Allows you to delete an item from a list. |
| LENGTH | Allows you to find out the length or number of items within a list. |
| THIS | Allows you to recurse through the Containment Model or retrieve information pertaining to the current container. |

Some examples of **eval** statements with keywords that are used in the AOC extensions are given below:

```
Occurred = eval (int,'(&Occurred) + 1')
CorrelatedId = eval (list type
long,'APPENDUNIQUE(&CorrelatedId,&&EventId)')
Description = eval (text,'CAT(`Now
subservient:`,&Description)')
```

# The eval Statement Datatypes

Almost any valid OQL or Stitcher language data type is allowed within the context of the **eval** statement, depending on where it is used—some datatypes are more applicable than others. The following table describes the different data types that are available.

*Table B-4 Table of data types available when constructing eval statements*

| Data type category | Data type syntax | Data type description |
|---|---|---|
| Character | **CHAR** [**ACTER**] | Stores fixed length characters. |
| | TEXT | Stores fixed length characters. |
| | **VARCHAR** ' (' n ') ' | Stores variable length characters. Length specified by positive integer n. |
| Numerical | **INT** [**EGER**] | Stores integer values. |
| | **INT** [**EGER**] **TYPE** <data_type> | Stores the conversion of data_type to integer datatype. |
| | SMALLINT | Stores an 8-bit integer. Used to save memory. |
| | FLOAT | Stores decimal values. |
| | LONG | Holds a 32-bit numerical value. |
| | LONGLONG | Holds a 64-bit numerical value. |

*Table B-4 Table of data types available when constructing eval statements (continued)*

| Data type category | Data type syntax | Data type description |
|---|---|---|
| Special | IPADDRESS | Specially designed for IP addresses. |
| | DATA | Holds opaque data, typically binary information. |
| Time | TIME | Holds a time. |
| List | **LIST TYPE** <data_type> | Holds a list of particular data_types. |
| Array | **ARRAY TYPE** <data_type> | Holds an array of a specified data_type. |
| Object | **OBJECT TYPE** <data_type> | Holds objects of particular data_types. |

# Other Recognized eval Statement Character Sequences

The character sequences below allow the inclusion of special characters such as ‘,’ ‘(‘ ‘)’ ‘&’ ‘$’ as text characters in evaluation clauses, where normally they would have special meanings.

*Table B-5 Table of eval statement character escape sequences*

| Character sequence | Interpretation |
|---|---|
| \\, | Allows you to escape the comma character, i.e., enables the OQL parser to treat it as literal text not having a special meaning. |
| \\( | Allows you to escape the open bracket character. |
| \\) | Allows you to escape the close bracket character. |
| \\$ | Allows you to escape the dollar symbol. |
| \\& | Allows you to escape the ampersand symbol. |

# Arithmetic operators in evaluation clauses

The following arithmetic operators are allowed in the evaluation clause.

*Table B-6 Operators used by evaluation clauses in descending order of precedence*

| Operator | Description | Associativity | Precedence |
|---|---|---|---|
| - | Negative sign | Non associative | 1 (Highest) |
| + | Addition | Left | 2 |
| - | Subtraction | | |
| * | Multiplication | Left | 3 (Lowest) |
| / | Division | | |

# Identifiers in eval Statements

As described in Table B-6 on page B-5, evaluation clauses can make use of three different types of identifier to compose an expression: variable references, database record references and object pointers. Brief descriptions and specifications of each one are described in the following sections.

# Variable references

A reference within the **eval** statements to any variable currently available in the system, i.e., global variables such as **$TIME**, **$SHELL**, **$HOSTNAME** and other environment variables, or Cisco-defined variables such as **$ObjectId**, **$EntityName**, can be done using the <*dollared*> identifier.

```
<dollared>::=
$DOLLARED_STRING
```

The ($) represents a variable, which could be a reference to a system or NMOS variable. The **$DOLLARED_STRING** has the following requirements:

- The dollared name must start with the dollar character ($);
- The second character must be a letter of the alphabet (upper or lower case);
- Subsequent characters must be alphanumeric characters; but
- There is no restriction on the name length.

Examples of references to variables are:

- $**TIME** // reference to the standard UNIX time variable.
- **$HOSTNAME** // reference to the machine running the process.
- **$SHELL** // reference to the shell type being used.
- **$ENV** // reference to the user shell environment.

# Database Record References

```
<ampersanded>::=
 &AMPERSAND_STRING
|&AMPERSAND_STRING <object_pointer_list>
```

The above syntax defines an (&) as a reference to a database record or an ampersanded target identifier list. Examples are shown below:

- &Severity
- &SuperClass
- &CreateTime

The **&AMPERSAND_STRING** identifier in the evaluation clause is a reference to any database object, usually a column from which a value can be obtained. The specifications of the **&AMPERSAND_STRING** identifier type are as follows:

- the ampersanded name must start with one or two ampersand characters (&) depending on the scope of the database record it is to reference;
- the character immediately after the final ampersand must be a letter of the alphabet (upper or lower case);
- subsequent characters must be alphanumeric; but
- there is no restriction on the name length.

A variant of the database record reference allows you to evaluate database records outside of the current scope using a double ampersand (&&), which was described in Chapter 2, "Object-oriented Principles and MWFM NMOS."

# Extraction of Record Values and Fields

There are two special functions that are used with the **eval** keyword to simplify the process of extracting record field names and record values, which is useful should you wish to transfer the contents of the current record into another database. These are listed below with an example showing the use of each in context:

- RECORD_NAMES(): extracts all field names of the current database record being considered.
- RECORD_VALUES(): extracts the values of all fields within the current database record being considered.

An example of these functions is given below:

```
insert into system.stateRules values
(
'modelCreate','',
'insert into kernel.activeModel
( eval(text,"RECORD_NAMES()" ))
values
( eval(text,"RECORD_VALUES()" ));',
1
);
```

The above example could be written in long form as follows:

```
insert into system.stateRules values
(
'modelCreate','',
'insert into kernel.activeModel
(
ObjectId,
EntityName,
Address,
Description,
EntityType,
ClassName,
EntityOID
)
values
(
eval(long, "&ObjectId"),
eval(text, "&EntityName"),
eval(list type text, "&Address"),
eval(text, "&Description"),
eval(int type entityTypes, "&EntityType"),
eval(text, "&ClassName"),
eval(text, "&EntityOID")
);',
1
);
```

The advantages of the first method are:

1. It is easier to understand because the meaning is immediately clear.
2. It is less prone to syntactical error.
3. You do not need to know about the fields within the incoming record because they are passed implicitly between the functions `RECORD_NAMES()` and `RECORD_VALUES()`.

# Pointers to objects: the target identifier

The arrow operator, `->`, represents a pointer to a named target. It extracts the value of the target from the database record which has been located (by, for example, recursion through the Containment Model using the **THIS** keyword).

```
<object_pointer_list>::=
->OBJIDENT_STRING { ' ' ->OBJIDENT_STRING }
```

The above syntax defines a list of object pointers as one or more space delimited pointers to an object value. An example is shown below:

```
->EntityName
->ObjectId
->EntityName ->ObjectId ->EventId
```

The specifications of the **->OBJIDENT_STRING** identifier type are as follows:

- The object identifier name must start with the arrow operator (->);
- The second character must be a letter of the alphabet (upper or lower case);
- Subsequent characters must be alphanumeric; but
- There is no restriction on the name length.

## Numbers in eval Clauses

The number types that are valid for use with OQL clauses are shown below:

```
INTNUM::=
 [0-9]+
```

The number type **INT_NUM** must be zero or any positive integer.

# General Syntax of the Evaluation Clause

The full syntax of the evaluation clause within the **eval** statement is given below:

```
<evaluation_clause>::=
<clause_expr> { ' ' <clause_expr> }
```

The above syntax implies that the evaluation clause can be a list of one or more space-delimited clause expressions.

```
<clause_expr>::=
   <clause_expr>  ' + ' <clause_expr>
| <clause_expr>  ' - '  <clause_expr>
| <clause_expr>  ' / '  <clause_expr>
| <clause_expr>  ' * '  <clause_expr>
| ' + ' <clause_expr>  // plus sign takes precedence above other
 // operators
| ' - ' <clause_expr>  // minus sign takes precedence above other
 // operators
| <statement>
| ' ( ' <clause_expr>  ' ) '
```

The above syntax shows that a clause expression is any valid algebraic combination of one or more evaluation statements.

## Evaluation Expression Statements

The following syntax defines the statements that constitute a valid evaluation expression.

```
<statement>::=
  <append_statement>
| <appendunique_statement>
| <delete_statement>
| <this_list>
| <sub_statement>
| <cat_statement>
| <length_statement>
| INT_NUM
```

# Variable and Database References

```
<sub_statement>::=
<ampersanded>
| <dollared>
```

The <*sub_statement*> is simply a reference to any variable or database record, as already described.

# Concatenating Lists

The evaluation clause syntax allows you the flexibility of concatenating data from different sources using the CAT keyword:

```
<cat_statement>::=
 CAT ' ( ' <cat_component> { ' , ' <cat_component> } ') '
<cat_component>::=
<ampersanded>
| <dollared>
| ABS_VAL
```

The items to be concatenated together can be any of the following:

- A reference to a database record
- A reference to a system, NMOS variable or Cisco Mobile Wireless Fault Mediator variable
- Empty, i.e., NULL
- An absolute value. An absolute value must be enclosed within single back quotes, e.g., `item`.

```
 CAT(`Now subservient:`,&Description )
```

# Appending Data to Lists

The APPEND keyword allows you to append data to the end of a list.

```
<append_statement>::=
APPEND ' ( ' &AMPERSAND_STRING ' , ' &AMPERSAND_STRING ') "
```

```
 APPEND ( &CorrelatedId , &EventId )
```

# Appending Unique Data to Lists

The APPENDUNIQUE keyword allows you to append data to a list if but only if it is unique to the list it is being appended to.

```
<appendunique_statement>::=
APPENDUNIQUE ' ( ' &AMPERSAND_STRING ' , ' &AMPERSAND_STRING ') '
```

```
 APPENDUNIQUE ( &CorrelatedId , &EventId )
```

## Deleting Data from a List

The DELETE keyword allows you to remove items from a list.

```
<delete_statement>::=
DELETE ' ( ' &AMPERSAND_STRING ' , ' &AMPERSAND_STRING ' ) '
```

```
 DELETE ( &CorrelatedId , &EventId )
```

## Finding the Number of Items in a List

Sometimes you may need to find out the length of a list (how many items it contains)—for example, in order to know how many times to execute a loop. This can be done using the LENGTH keyword:

```
<length_statement>::=
LENGTH ' ( ' &AMPERSAND_STRING ' ) '
```

The returned value from LENGTH is then coerced into the integer data type.

```
LENGTH ( &CorrelatedId ) // raw syntax definition.
eval (int, 'LENGTH( &CorrelatedId))' // Use within an eval
statement.
```

# The Containment Model Evaluators

The Containment Model evaluators are only applicable when the NMOS has constructed a Containment Model, i.e., after a discovery has been sent to MODEL. These evaluators therefore limit themselves to the database records found in MODEL's master database.

In order to be able to move around and recurse through the different levels of the Containment Model, the evaluation clause has a special expression—the **THIS** keyword expression.

## THIS Keyword

The use of the THIS keyword is given by the syntax definition below:

```
<this_list>::=
  <this_statement>
```

The above syntax defines two alternate ways in which you can make use of the **THIS** keyword. You will be introduced to each separately; firstly the <*this_statement*> form is discussed below:

```
<this_statement>::=
  THIS ' ( '
<num_or_constant> ' , '
<num_or_constant> ' , '
<num_or_constant>
' ) ' ->OBJIDENT_STRING
| THIS ->OBJIDENT_STRING
```

The above defines the syntax for the use of the first form of the **THIS** keyword.

```
<num_or_constant>::=
INT_NUM
```

```
| <dollared>
```

The <*num_or_constant*> is only applicable to the **THIS** keyword and can either be an integer value or a variable that evaluates itself to an integer. The format of the <*this_statement*> as implied from the syntax is shown below:

- **THIS** ( A , B , FLAG ) ->EntityName
- **THIS** ->EntityName

The second example extracts the EntityName value for the current MODEL entity.

In its basic format, the <*this_statement*> is made up of three values: A, B, and FLAG. The value A signifies how many levels to traverse ***up*** through the Containment Model, while B represents how many levels to traverse ***down*** the Containment Model.

The FLAG denotes whether all entities encountered while traversing the different levels of the containment model should be accumulated into a list, or if only the final destination of the traversal should be considered.

## Values for A, B, and FLAG.

A, B, and FLAG are flexible enough to accept either integer constants or variables (<*dollared*>) that evaluate to integer constants as valid inputs. There are, however, three special variables that can be used, which are named and discussed below:

1. $PHYSICAL—When used with THIS, the traversal will terminate at the top of the Containment Model.
2. $INCLUDERECURSED—When used with THIS, all entities encountered in the Containment Model during the traversal of the levels will be included and accumulated together as a list.
3. $EXCLUDERECURSED—all entities encountered during the traversal of the Containment Model will be excluded. Only the final entity at the end of the traversal will be considered.

An example of the use of some of the keywords and variables are shown below.

```
eval (list type text, 'THIS(0, 1, $INCLUDERECURSED) ->
EntityName')
```

# The target identifier (-> OBJIDENT_STRING)

While the **THIS** keyword allows recursion and movement through different levels of the Containment Model, you will need to be able to specify the column name of the Containment Model element(s) to extract. This is done by specifying the target identifier using the arrow operator (->), as seen in the example above.

It follows logically that, in view of the fact that the target identifier is used within the context of the Containment Model, the targets should be references to column names within the tables of the MODEL databases (where all information for every network entity is detailed). Hence, examples of target identifiers are:

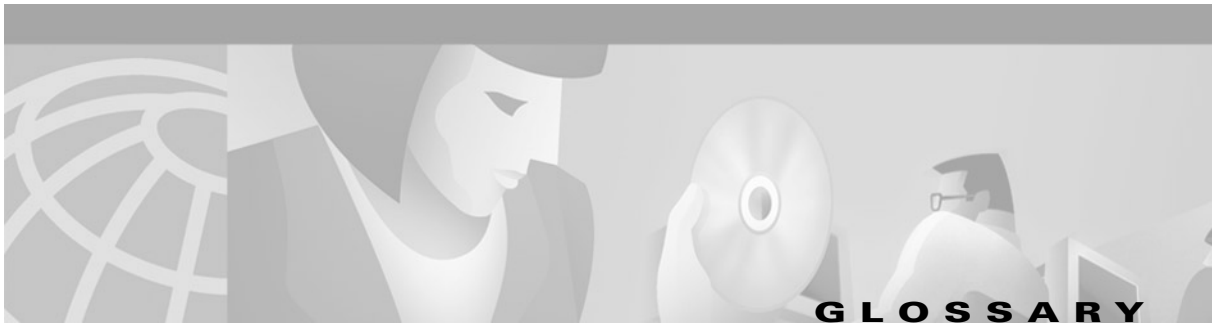
- ObjectId
- EntityName
- Address
- Description

- EntityType
- ClassName
- EntityOID
- RelatedTo

For a list of all tables and column names within the MODEL databases, see the chapter on MODEL databases in the *Cisco Mobile Wireless Fault Mediator 2.1 - Topology and Platform Modeling Reference Guide*.

# Conclusion

This appendix has introduced you to the full functionality and syntax of the **eval** statement, which can be used within the AOC extensions, OQL and the Stitcher language to evaluate database records and variables or coerce them to specified data types, and most importantly allows you to explore the constituent elements and containers of the Containment Model.



# GLOSSARY

## A

**Accounting** — See FCAPS.

**Action firing condition** — The final logical test to determine whether a specified action is executed. The action firing condition is applied to the actions triggers.

**Actions** — Outputs from the method conclusion, these are processes that enable the event records in the AMOS Events database to be manipulated once the respective action firing conditions have been satisfied. Create, change and delete are actions.

**Actions stage** — The stage which determines how the event records in the AMOS Events database are manipulated. The action stage consists of the actions triggers, action firing condition and actions.

**Actions triggers** — A list of all events and alerts that have transpired on all devices (and entities contained within the devices) specified by applying the control location in the network topology model and the "run in containers" component of the Containment Model. The actions triggers are the input to the actions stage.

**Alert** — A network event that requires the attention of a network administrator. These are deemed to be significant enough to be assigned an alert status, based on a company's network management policy, with the level of severity indicated by a designated color, according to industry standards.

**AMOS** — Active Managed Object Store. The event correlation engine of Cisco Mobile Wireless Fault Mediator.

**AMOS Class database** — Contains information pertaining to an instance's class—instantiation information, parent class information, data dictionary details and the AOC extensions.

**AMOS Entity database** — Contains all information pertaining to network entities, their containment and connections.

**AMOS Events database** — Contains a log of all information pertaining to event records generated MONITOR.

**AOC** — Active Object Class. A dynamic feature of the MWFM NMOS based on the concept of object class files and object-orientation. AOCs are considered "active" since there is no operator input. In this case, the AOC tells the system what action to undertake on the object upon instantiation based on the properties, behaviors and management policies contained it contains. Implementation of the AOC properties occurs as soon as the object has been detected in the network. For reference, a "passive" class can be defined as a situation where the system needs to be told that an object belongs to a particular AOC before it instigates the properties, behaviors and management policies associated with the object.

**AOC browser** — Active Object Class browser. A Graphical User Interface (GUI) standalone application that allows the AOCs to be accessed, edited and manipulated.

**AOC extensions** — See Extensions.

**Asynchronous** Processes that proceed independently of each other until one process needs to "interrupt" the other process with a request

**AUTH** A component of MWFM NMOS. AUTH (representing the authentication engine) monitors actions of MWFM NMOS users to ensure that any action they attempt to take in relation to network devices, events and alerts is one for which they have the necessary security privileges.

## B

**Base generated event** The information present in the event record that is sent to AMOS in the event stream unless it is overridden by one of the event override lists.

**Broadcast** A message sent over a network to all members (or subjects)—rather than specific members (or subjects)—of a group such as a department or enterprise.

## C

**Change** An action, see Actions.

**Child Class** A class directly below another in a hierarchical structure; a class that inherits from another class.

**Class** In object-orientated programming, a class is a template definition of the methods and variables in a particular kind of object. An object is defined as a specific instance of a class; it contains real values instead of variables.

**CLASS** The MWFM NMOS component that is responsible for managing and distributing information contained within the Active Object Classes to other MWFM NMOS processes and applications that may require class-based information.

**Component** The constituent element of a system. For instance, MWFM NMOS components include, CLASS, DISCO, MONITOR, MODEL, DIST, CONTROL and STORE.

**Configuration Management** See FCAPS.

**Connectivity** The manner in which devices that form a network are connected to each other and how they interact.

**Containers** Objects that can "hold" a collection of other objects or entities, some of which may themselves be containers.

**Containment Model** A hierarchical structure that enables MWFM NMOS to establish a relationship between physical and logical network entities or objects. The Containment Model enables potentially unrelated physical and logical entities or objects to be linked simply because they are held in the same repository or "container", thus allowing MWFM NMOS to detect them and incorporate them in to the network topology.

**Control location** Used to scope the method to particular network objects within the Network Topology Model.

**Correlation engine** See Event correlation engine.

**Create** An action, see Actions.

**CTRL** The MWFM NMOS component responsible for controlling and managing all other components in the MWFM NMOS suite.

## D

**Data dictionary** Similar to a library. It contains all of the ASN.1 descriptions of all of the data types used by the Active Object Class.

**Delete** An action, see Actions.

**Details Agent** The Agent responsible for the retrieval of basic information about devices discovered in the network by the Finders.

**Devices** Any part of the computer other than the CPU and the working memory. In networking terms, it refers to the different entities that make up the network, e.g. desktops, switches, routers, hubs, printers, interfaces, etc.

**DISCO** The key component in MWFM NMOS. It runs as a two-stage process. The first stage of the process finds devices on the network and then explores the connectivity between them to discover all the possible routes for data. The second stage of the process correlates all this information and builds up a complete map of the network topology. DISCO dynamically updates this map as connections are lost, regained or newly discovered.

**Discovery** The process of discovering which devices are located on your network and determining from those devices an accurate picture of how the devices interconnect with one another and the server(s), ultimately building a map referred to as the network topology.

**Discovery Agent** Processes that retrieve information about devices on the network. They are also capable of reporting the existence of new devices by virtue of the connections found to other devices.

**DIST** A component of MWFM NMOS. DIST is a server application which distributes the NMOS to multiple hierarchical domains. DIST is also the component that enables MWFM NMOS to integrate with third-party applications such as RDBMS and help desk systems.

## E

**Event** An event is any change of state of any device on the network.

**Event construction list** A list of attributes used to construct the base generated event.

**Event correlation** Comparison of events on the network with the event correlation methods stored in the AOCs to determine whether a series of events means that a particular device on the network has failed or is not responding.

**Event correlation engine** An engine that compares events on the network with the event methods stored in the AOCs to determine whether a series of events mean that a particular device on the network has failed or is not responding.

**Event correlation method(s)** Policies that are added to the AOCs to define how to analyze events that occur on the network and determine their severity level and whether they require user attention.

**Event management** The practice of processing and correlating network events.

**Event methods** See Event correlation methods

**Event override lists** Threshlist, Pollfaillist, Restorelist. These allow the network administrator to override the column names in the base generated event with custom values.

**Event stream** Each event that is generated by MONITOR as a result of the polling process is broadcast to extensions such as Cisco Mobile Wireless Fault Mediator. This series of events is known as the event stream.

**Extensions** Additions to the AOCs made by the respective network management applications, which extend the NMOS' functionality. The extensions associated with Cisco Mobile Wireless Fault Mediator are: the poll definitions, the event correlation methods, and the menu methods.

## B

**Fault management** See FCAPS.

**FCAPS** FCAPS is a conceptual model of network management developed by the International Organization for Standardization to provide a clear focus and strategy for running a network safely and efficiently. The letters are an acronym representing the five areas a network management policy needs to consider. Fault management is about identifying faults on a network quickly and efficiently. Configuration management tracks the various versions of hardware and software on a network, ensuring lack of conflict (or reducing it to an acceptable level). Accounting applications monitor individual uses of a network for quota calculations and billing. Performance management software identifies areas of a network that are overloaded, allowing efficiency to be improved, and Security software is concerned with controlling access to network resources and preventing unauthorized use of or tampering with the network.

**FCAPS model** See FCAPS.

**Finder(s)** Processes that discover the existence of devices on the network; however, they do not report connections like the Agents.

**Firing policy** Dictates whether the event correlation method conclusion will be fired or not.

## G

**GUI** Graphical User Interface. Any system that allows you to control a computer through graphical elements (icons, menus, dialogue boxes and so on) displayed on screen and selectable by pointing and clicking.

## H

**Helper(s)** Applications that go into devices and assist the Agents in retrieving connectivity information. They have no understanding of the information.

**Hierarchy** A structure that ranks components into levels of subordination according to a defined criteria. In object-orientation the relationship between the parent class and child class forms a hierarchy.

## I

**Impact Topology** The activated scratch topology after discovery that is used by the other processes for root-cause analysis.

**Inheritance** In object-orientation, the ability to derive new classes from existing classes. A derived class ("child class") inherits the instance variables and methods of the base class ("parent class"), and may add new instance variables and methods. New methods may be defined with the same names as those in the base class, in which case they override the original one.

**Instance** In object-oriented terms, this refers to a specific physical manifestation of a class. See also class, instance, instantiation, and object.

**Instantiation** In object-orientation, the act of creating a new instance of an object class and applying the properties, behavior, and management policies associated with that class of object.

**Internal test flag** A flag to indicate whether a Threshold condition has been evaluated true on the previous evaluation. Once set, the flag can only be cleared by a restore condition.

**IP** Internet Protocol. The underlying network layer protocol for routing packets over the Internet and other TCP/IP-based networks, used to establish a virtual connection between a source and destination. See also TCP/IP.

**IP address** Internet Protocol address (TCP/IP address). A 32-bit host address defined by the Internet Protocol. It is usually represented in dotted decimal notation. For example, 207.168.124.255.

**ISO** International Organization for Standardization. A worldwide federation of national standards bodies. The OSI reference model and the FCAPS model are maintained by this federation.

## M

**Menu methods** Extensions in the AOC that enable menu actions to be associated with objects, so a user can interact directly with those objects.

**Method chaining** A process which allows a series of simple event correlation methods to be combined to form more complex event correlation methods.

**Method conclusion** Output from an event correlation method; the components include the actions (create, change and delete), the directive command, and method control command.

**Method control command** Allows a virtual daemon to be associated with devices on the network.

**Method firing condition** The first section of an event correlation method; a logical test to determine whether the rest of the event correlation method is executed (fired) or not.

**Method triggers** A means of initiating an event correlation method. This can be achieved by events in the event stream from MONITOR, a timer in the method itself, or events output from other event correlation methods.

**MIB** Managed Information Base. A formal description of a set of network objects that can be managed using the Simple Network Management Protocol (SNMP).

**MODEL** A component of MWFM NMOS. MODEL takes information from the DISCO component about the devices and connections that make up a network and stores that information for reference. MODEL only stores this information; it does not process the information in any way.

**MONITOR** A component of MWFM NMOS. MONITOR continually (or at intervals chosen by the network operator) polls devices on the network to find out whether their connection is still active or if that device is no longer available. It interacts with the MODEL component to find out what devices it needs to be looking for. If MONITOR finds that a device recorded in the network topology of MODEL has become unavailable, it passes on that information to the relevant components of NMOS or Cisco Mobile Wireless Fault Mediator.

**Multicast** A point-to-point means of transmitting data to a selected group of users, all at the same time. Standard TCP connections are unicast, that is, single sender and single recipient - to send a data stream to 10 viewers would require 10 separate data streams. An alternative is broadcast, which is single point to every recipient but the traffic goes to more viewers than have requested it, hence affecting their links into the network. Multicast is point to multipoint, which means a single stream from a server goes simultaneously to the set of recipients that it is intended for.

## N

**NMOS** Network Management Operating System. The base architecture provided by MWFM NMOS. The operating system provides a central platform which is able to integrate separate network management applications, such as Cisco Mobile Wireless Fault Mediator.

## O

**Object** In object-orientation, a unique instance of a data structure defined according to the template provided by its class. Each object has its own values for the variables belonging to its class and can respond to the messages (methods) defined by its class.

**OQL** Object Query Language. An interactive programming language, similar to SQL, used for getting information from and updating a database. However, unlike SQL, OQL is object-based and designed specifically around the operational needs of the MWFM discovery process. It has the ability to support object referencing within tables, making it possible to have objects nested within objects. It also contains a convention for describing how to reference queries and how to store them.

## P

**Packet** The unit of data sent across a network.

**Parent Class** The class directly above another class; from which attributes and management policies are inherited.

**Performance Management** See FCAPS.

**Ping** Packet Internet Groper. A protocol for testing whether a particular computer is connected to the Internet by sending a packet to its IP address and waiting for a response. The origins of the name come from the terminology used in submarine SONAR, where a sound signal (called a “ping”) is broadcast, and surrounding objects are revealed by their reflections of the sound.

**Ping Agent** A synchronous Polling Agent which ensures a device is still contactable by sending a ping.

**Poll** See Polling.

**Poll definitions** These determine how MONITOR and the Polling Agents poll the network; for instance, how often an object is polled, the type of Polling Agent employed to do the polling, the frequency of the polling, what information is sought during the polling process, what traps are set, etc. More importantly, the poll definitions determine what events are generated when a device undergoes a state change. These events or the event stream as it is known, represent the input to AMOS.

**Poll fail condition** The condition that occurs when a device does not respond to polling attempts within the timeout period, i.e. a device is unreachable.

**Poll list** A list of attributes available for constructing a poll definition within an AOC.

**Poll strategy** The selection of an AOC file that determines how, when and where the Polling Agents will interrogate the network for information pertaining to a device’s state (availability) or aspects associated with its performance.

**Poll strategy list** A list of attributes associated with the polling strategy.

**Polling** The process of periodically querying a device so the system can elicit the device’s state (availability) or some other information.

**Polling Agents** Processes which monitor the status of network devices after the discovery process has generated a network topology.

**Polling methodologies** See Poll definitions.

**Protocol** A set of standards that is designed to enable diverse computers to connect together and exchange information with one another.

## R

**Restore condition** A condition that occurs when the first evaluation of a Threshold condition is false after the internal test flag was set to fail because of a Threshold success.

**riv_auth** See AUTH.

**riv_class** See CLASS.

**riv_ctrl** See CTRL.

**riv_dist** See DIST.

**riv_f_amos** See AMOS.

**riv_model** See MODEL

**riv_monitor** See MONITOR

**riv_oql** See OQL.

**riv_store** See STORE

**Cisco Mobile Wireless Fault Mediator** Cisco's solution to the Fault Management area identified in the ISO FCAPS Model. Cisco Mobile Wireless Fault Mediator is an event processing and correlation engine; it essentially correlates a stream of network events with event correlation methods and determines which events need to be upgraded to alert status.

**Root Class** The class at the top or beginning of a class hierarchical structure.

**Root-cause** An alert which, frequently, will cause other (sympathetic) alerts to arise in the network management application because it has devices which are dependent upon it for their own connections. Correcting this problem will fix the others, and hence this alert is the root of the trouble.

**Router** Referred to as an intermediary device on a communications network that expedites message delivery. On a single network linking many computers through a mesh of possible connections, a router receives transmitted messages and forwards them to their correct destinations over the most efficient available route. On an interconnected set of local area networks (LANs) using the same communications protocols, a router serves the somewhat different function of acting as a link between LANs, enabling messages to be sent from one to another.

## S

**Security Management** See FCAPS.

**Server** On a local area network (LAN), a computer that runs administrative software to control access to the network and its resources, such as printers and disk drives, providing these as resources to other computers that function as workstations on the network. 2. On the Internet or other wide area network, a computer or program that responds to requests from a client. For example, a file server may contain an archive of data or program files; when a client submits a request for a file the server transfers a copy of the file to the client.

**SNMP** Simple Network Management Protocol. A management protocol for TCP/IP networks.

**SNMP Agent** A synchronous Polling Agent used to collect information from devices on the network that use the SNMP protocol.

**Stitcher(s)** Responsible for the transfer of information between applications after processing in accordance with predefined stitching rules.

**STORE** A component of MWFM NMOS. Whenever two components of MWFM NMOS interact, sending messages, requests or data, STORE will record the fact that a transaction took place between these two components and what that transaction was. It acts as both a log of actions that have occurred within MWFM NMOS and as a backup in the event that problems occur.

**Sub Class** See Child Class.

**Super Class** See Parent Class.

**Switch** In network communications, a computer or electromechanical device that controls routing and operation of a signal path.

**Synchronous** A process that transmits at regular times or intervals; as a transmission is received, a response is returned indicating success or the need to resend the information.

**Syslog Agent** An asynchronous Polling Agent which monitors system log files by parsing them at intervals looking for updated information.

## T

**TCP/IP** Transmission Control Protocol / Internet Protocol. This is the basic communication protocol used for the Internet and other networks. TCP/IP has two layers; Transmission Control Protocol divides a message or file into small packets of data to be reassembled by the destination device's TCP layer; the lower layer, Internet Protocol, handles the addressing of each packet so it reaches the destination. See also IP.

**Threshold** An attribute of the poll list used by the SNMP, Trap and Syslog Agents as a basis for attributes to poll for and forms the basis of a logical test to assess whether an event is generated as a result of polling.

**Threshold condition** When the poll Threshold succeeds, i.e. evaluates true an event with attributes specifying this fact is sent to AMOS.

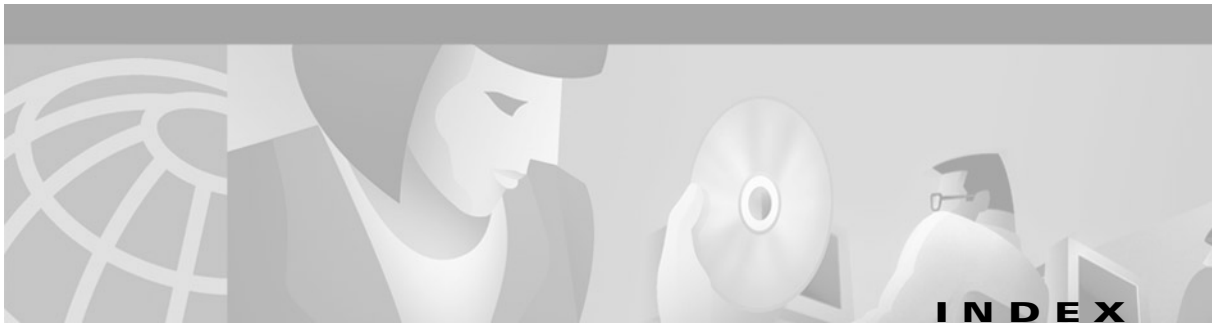
**Topology** The configuration formed by the connections between devices on a local area network (LAN) or between two or more LANs.

**Trap** A program interrupt, usually an interrupt caused by some exceptional situation in the user program. In most cases, the OS performs some action, and then returns control to the program.

**Trap Agent** An asynchronous Polling Agent that listens for traps defined in the poll definitions.

## V

**Virtual daemons** Processes which enable the actions stage of the method conclusion of an event correlation method to run on objects specified by the control location without having to re-run the entire event correlation method.



# INDEX

## A



## B



## C



## D

## E



## F



## H



## I



## M



## N

## O



## P



## R

## S



## T