# Universal Event Monitor

## User Guide

Universal Products

Version 3.2.0

# Universal Event Monitor

## User Guide

## Universal Products 3.2.0

| Document Name | Universal Event Monitor 3.2.0 User Guide | | | | |
|---|---|---|---|---|---|
| Document ID | uem-user-3203 | | | | |
| Products | z/OS | UNIX | Windows | OS/400 | HP NonStop |
| Universal Event Monitor Manager | √ | √ | √ | | |
| Universal Event Monitor Server | | √ | √ | | |
| UEMLoad | | √ | √ | | |

# Stonebranch Documentation Policy

This document contains proprietary information that is protected by copyright. All rights reserved. No part of this publication may be reproduced, transmitted or translated in any form or language or by any means, electronic or mechanical, including photocopy, recording, or any information storage and retrieval system, without permission, in writing, from the publisher. Requests for permission to make copies of any part of this publication should be mailed to:

Stonebranch, Inc.
950 North Point Parkway, Suite 200
Alpharetta, GA 30005 USA
Tel:   (678) 366-7887
Fax:  (678) 366-7717

Stonebranch, Inc.® makes no warranty, express or implied, of any kind whatsoever, including any warranty of merchantability or fitness for a particular purpose or use.

The information in this documentation is subject to change without notice.

Stonebranch shall not be liable for any errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this document.

All products mentioned herein are or may be trademarks of their respective owners.

© 2003-2010 by Stonebranch, Inc.

All rights reserved.

SAP Certified Integration

# Summary of Changes

**Changes for Universal Event Monitor 3.2.0 User Guide
(uem-user-3203)
September 8, 2009**

- Added the following examples in Section A.2 UEM Manager for z/OS Examples of Appendix A Examples:
  - Continuation Character "-" in z/OS Handler Script
  - Continuation Character "+" in z/OS Handler Script
  - Continuation Characters "-" and "+" in z/OS Handler Script
  - Starting a UEM (Event-Driven) Server
  - Refreshing a UEM Server (Event-Driven)

**Changes for Universal Event Monitor 3.2.0 User Guide (uem-user-3202)
December 17, 2008**

- Added Updating the Universal Event Monitor Server ACL Entries in Section 8.5.5 Universal Access Control List of Chapter 8 Universal Event Monitor Server for Windows.

**Changes for Universal Event Monitor 3.2.0 User Guide (uem-user-3201)
September 5, 2008**

- Added toll-free telephone number for North America in Appendix D Customer Support.
- Added UACL Entries section in Section 8.5.5 Universal Access Control List.
- Added UACL Entries section in Section 9.5.5 Universal Access Control List.

### Changes for Universal Event Monitor 3.2.0 User Guide (uem-user-320) May 16, 2008

**Universal Event Monitor 3.2.0**

- Added support for the following features:
  - UCM and UDM Handlers
    UEM handlers now include Universal Command and Universal Data Mover handler types providing direct and seamless integration with the Universal Product infrastructure.
  - UEMLOAD Input
    The UEMLOAD input file can now be read from standard input greatly simplifying the execution of UEMLOAD with Universal Command.
- Added Section 2.4 Universal Configuration Manager.
- Added the following configuration options for Universal Event Monitor Manager:
  - BIF_DIRECTORY (UNIX only)
  - CONNECT_TIMEOUT
  - DNS_EXPAND
  - HANDLER_TYPE
  - HOST_SELECTION
  - INSTALLATION_DIRECTORY (UNIX and Windows only)
  - NLS_DIRECTORY (UNIX and Windows only)
  - PLF_DIRECTORY (UNIX only)
  - SYSTEM_ID (z/OS only)
- Modified the following configuration option for Universal Event Monitor Manager:
  - REMOTE_HOST
- Added the following configuration option for Universal Event Monitor Server:
  - NLS_DIRECTORY (UNIX and Windows only)
- Added the following configuration options for UEMLoad:
  - HANDLER_TYPE
  - OPTIONS

### Changes for Universal Event Monitor 3.1.1 User Guide (uem-user-31111) February 28, 2007

- Added List of Figures and List of Tables.
- Added Appendix C Customer Support.

**Universal Event Monitor 3.1.1.6**

- Changed name of DD statement in MESSAGE_LEVEL option of Section 3.1.3.3 Command Options Reference of Section 3.1Universal Event Monitor Manager for z/OS.
- Added Configuration File Keyword to:

- POLLING_INTERVAL in Section 3.1.3.3 Command Options Reference of Section 3.1 Universal Event Monitor Manager for z/OS.
- POLLING_INTERVAL in Section 3.3.2.4 Command Options Reference of Section 3.3 Universal Event Monitor Manager for UNIX.
- Deleted information for option value of NO in:
  - RENAME_FILE in Section 4.2 Universal Event Monitor Server for Windows
  - RENAME_FILE in Section 4.3 Universal Event Monitor Server for UNIX
  - RENAME_FILE in Section 5.3 UEMLoad Utility for Windows and UNIX.
- Added INSTALLATION_DIRECTORY option in Section 4.3.3.2 Configuration Options Reference of Section 4.3 Universal Event Monitor Server for UNIX.
- Changed values of the Active Flag parameter from True/False to Yes/No in Section 5.2.2.1 Event Definition Parameters.
- Added environment variables to Table 13  Environment Variables Set by UEM in Appendix B .

## Changes for Universal Event Monitor Release 3.1.1.5
## January 30, 2006

- Added INTERACT_WITH_DESKTOP and TRACE_DIRECTORY Server configuration options.
- Added examples to demonstrate the use of wildcards when the -list or -export option is specified with the Universal Event Monitor Load utility.
- Documented new default behavior for Universal Event Monitor Load utility whereby all event definition and event handler records are returned if the -event_id and -handler_id parameters are omitted when the -list or -export option is specified.
- Added Appendix A to describe methods for capturing event handler process output using an event definition's handler_options parameter.
- Added Appendix B to list environment variables which are set by UEM Server and made available to event handler processes.

## Changes for Universal Event Monitor Release 3.1.1.0
## April 30, 2005

- Completed Universal Event Monitor Server chapter
- Added Universal Event Monitor Manager chapter

## Changes for Universal Event Monitor Release 3.1.0.0
## October, 2004

- New document.

# Contents

# List of Figures

# List of Tables

# Preface

## Document Structure

This document is written using specific conventions for text formatting and according to a specific document structure in order to make it as useful as possible for the largest audience. The following sections describe the document formatting conventions and organization.

## Format

Starting with the Universal Products 3.2.0 release, the Universal Event Monitor User Guide has been reformatted and restructured.

Most importantly, links to detailed information in a companion document, the Universal Event Monitor Reference Guide, have been created in the user guide.

In order for the links between these documents to work correctly:

- Place the documents in the same folder.
- In Adobe Reader / Adobe Acrobat, de-select **Open cross-document link in same window** in the **General** category of your **Preferences** dialog (selected from the **Edit** menu).

# Conventions

Specific text formatting conventions are used within this document to represent different information. The following conventions are used.

## Typeface and Fonts

`This Font` identifies specific names of different types of information, such as file names or directories (for example, `\abc\123\help.txt`).

## Command Line Syntax Diagrams

Command line syntax diagrams use the following conventions:

| Convention | Description |
|---|---|
| `bold monospace font` | Specifies values to be typed verbatim, such as file / data set names. |
| `italic monospace font` | Specifies values to be supplied by the user. |
| `[]` | Encloses configuration options or values that are optional. |
| `{}` | Encloses configuration options or values of which one must be chosen. |
| `|` | Separates a list of possible choices. |
| `...` | Specifies that the previous item may be repeated one or more times. |
| **BOLD UPPER CASE** | Specifies a group of options or values that are defined elsewhere. |

Table P.1  Command Syntax

## Operating System-Specific Text

Most of this document describes the product in the context of all supported operating systems. At times, it is necessary to refer to operating system-specific information. This information is introduced with a special header, which is followed by the operating system-specific text in a different font size from the normal text.

**z/OS**

This text pertains specifically to the z/OS line of operating systems.

This text resumes the information pertaining to all operating systems.

## Tips from the Stoneman



Look to the Stoneman for suggestions
or for any other information
that requires special attention.

**Stoneman's Tip**

# Vendor References

References are made throughout this document to a variety of vendor operating systems. Stonebranch, Inc. attempts to use the most current product names when referencing vendor software.

The following names are used within this document:

- **z/OS** is synonymous with IBM z/OS and IBM OS/390 line of operating systems.
- **Windows** is synonymous with Microsoft's Windows 2000 / 2003 / 2008, Windows XP, Windows Vista, and Windows 7 lines of operating systems. Any differences between the different systems will be noted.
- **UNIX** is synonymous with operating systems based on AT&T and BSD origins and the Linux operating system.

Note: These names do not imply software support in any manner. For a detailed list of supported operating systems, see the Universal Products 3.2.0 Installation Guide.

# UEM Terminology

A complete description of the features and capabilities of Universal Event Monitor requires the use of specific terms, which appear throughout this document. These terms, along with their meaning, are listed below for easy reference.

***Demand-driven Server*** – A UEM Server process that is started by a Universal Broker at the request of a UEM Manager or the UEMLoad utility. In this situation, the UEM Server is said to have started upon demand. When a demand-driven UEM Server is started by a UEM Manager, it will run only until monitoring of a single event is completed. When a demand-driven Server is started by the UEMLoad utility, it will only run until the requested database operation is finished. (See also ***Event-driven Server***.)

***Event definition*** – The parameters that define a system occurrence to UEM. These parameters may come from the command line of a UEM Manager or from a record stored in the event definition database.

***Event handler*** – A response to an event occurrence, an event handler specifies the action to take when a system event monitored by UEM satisfies certain conditions. The action taken may be a system command or a script.

***Event-driven Server*** – A UEM Server process that is started automatically by a Universal Broker, without a request from a client. An event-driven Server monitors only those system events that have been assigned to it via an event definition record. (See also ***Demand-driven Server***.)

***Event occurrence*** – Each instance of a system event detected by Universal Event Monitor. Any event monitored by a given UEM Server may be defined so that any number of distinct system occurrences may satisfy the event criteria. Each system occurrence that satisfies the event criteria will be detected by UEM as an event occurrence. It follows, then, that each event monitored by UEM will result in the detection of 0 (zero) or more event occurrences.

***Expired*** – Describes an event state set by UEM. It also refers to the event handler that is executed whenever this state is set.

Unlike the other two states that can be set by UEM, ***Triggered*** and ***Rejected***, `expired` refers to an event definition itself, and not to a specific event occurrence.

Only one test is conducted before an event is set to an `expired` state. If the event becomes inactive and no occurrence of that event was detected by UEM, it will be set to an expired state.

The event definition may specify an event handler to execute when an event expires, but the assignment of an expired handler is optional. The state of an event does not depend upon whether this assignment was made.

***Handler process*** – The system command or set of script statements that is executed by UEM on behalf of a given event handler.

*Rejected* – Describes the state of an event occurrence. It also refers to the event handler that is executed whenever an event occurrence enters this state.

(See also *Expired* and *Triggered*.)

If UEM determines that an event occurrence has not completed within the allotted time (that is, before the event is scheduled to become inactive), that occurrence is set to a `rejected` state.

If the event definition specifies an event handler to execute when an event occurrence is rejected, that handler is referred to as the rejected handler. As with a triggered handler, the assignment of a rejected handler is optional, and the state of the event occurrence does not depend upon whether this assignment is made.

*Tracking* – Describes the state of a system event detected and monitored by UEM that has not met the completion criteria specified in the event definition.

*Triggered* – Describes the state of an event occurrence. It also refers to the event handler that is executed whenever an event occurrence enters this state.

(See also *Expired* and *Rejected*.)

When an event occurrence is detected, UEM performs tests on that occurrence, based on criteria specified in the event definition, to determine when it completes. When UEM determines that an event occurrence is complete, it sets that occurrence to a triggered state.

The event definition may also specify an event handler to execute when an occurrence of an event becomes triggered. This assignment of a triggered handler is optional, and the state of the event occurrence does not depend upon whether this assignment is made.

# Document Organization

The document is organized into the following chapters:

- Overview (Chapter 1)
  General architectural and functional overview of Universal Event Monitor.
- Features (Chapter 2)
  Information about configuration, protocol, databases, Universal Access Control List, message and audit facilities.
- Universal Event Monitor Manager (Chapter 3)
  General description of the UEM Manager functionality.
- Universal Event Monitor Manager for z/OS (Chapter 4)
  Description of the UEM Manager, plus procedures and options specific for z/OS operating systems.
- Universal Event Monitor Manager for Windows (Chapter 5)
  Description of the UEM Manager, plus procedures and options specific for Windows operating systems.
- Universal Event Monitor Manager for UNIX (Chapter 6)
  Description of the UEM Manager, plus procedures and options specific for UNIX operating systems.
- Universal Event Monitor Server (Chapter 7)
  Description of the UEM Server, plus procedures and options specific for Windows and UNIX platforms
- Universal Event Monitor Server for Windows (Chapter 8)
  Description of the UEM Server, plus procedures and options specific for Windows operating systems.
- Universal Event Monitor Server for UNIX (Chapter 9)
  Description of the UEM Server, plus procedures and options specific for UNIX operating systems.
- Universal Event Monitor Load Utility (Chapter 10)
  General description of the UEMLoad Utility.
- UEMLoad Utility for Windows (Chapter 11)
  Description of the UEMLoad Utility, plus procedures and options specific to Windows operating systems.
- UEMLoad Utility for UNIX (Chapter 12)
  Description of the UEMLoad Utility, plus procedures and options specific to UNIX operating systems.
- Examples (Appendix A)
  Operating system-specific examples that demonstrate the use of UEM.
- Standard I/O Redirection and Event Handler Processes (Appendix B)
- Environment Variables Set by Universal Event Monitor (Appendix C)
- Customer Support (Appendix D)
  Customer support contact information for Universal Event Monitor.

# Chapter 1
# Overview

## 1.1 Introduction

Universal Event Monitor (UEM) provides a consistent, platform-independent means of monitoring one or more local or remote system events. It also can execute a system command or script based on the outcome of the events that it monitors.

One or more system events can be monitored by UEM at any given time. UEM requires only that any event of interest be presented to it in a well-defined manner.

The methods available for defining an event and its associated actions, and then presenting those definitions to UEM, are described in the following sections.

# 1.2 UEM Functionality

Use the Universal Event Monitor (UEM) Manager to monitor a single local or remote system event.

The UEM Manager (`uem`) may provide all of the parameters necessary to define a system event, or it may specify the ID of a database record that contains the event definition. In either case, the UEM Manager passes the event definition to a local or remote UEM Server (`uemsrv`), which uses that information to look for an occurrence of the event and test for its completion.

The UEM Manager may also provide all of the parameters necessary to define an event handler to the UEM Server, or it may specify the ID of a database record that contains the event handler.  An event handler is a command or script that UEM Server executes, based on the outcome of the event occurrence.

A UEM Server may monitor several local system events simultaneously using records stored in its event definition database. An event-driven UEM Server executes in this manner. An event-driven UEM Server does not require a UEM Manager to initiate a monitoring request, and you may configure it to start automatically whenever the local Universal Broker starts. During start-up, an event-driven UEM Server retrieves a list of its assigned event definitions from the local Universal Broker. UEM Server monitors each event until it is no longer active, or until the event-driven Server ends.

The UEMLoad utility (`uemload`) enables you to add event definition and event handler records to their respective databases

UEMLoad handles all event definition and event handler database management tasks, including adds, updates, deletes, and lists / exports. UEMLoad forwards a database request to a UEM Server, which validates the information. The UEM Server then sends a request to a local Universal Broker to apply the requested operation to the appropriate UEM database file.

Figure 1.1, below, illustrates the interaction of the various components that make up Universal Event Monitor.
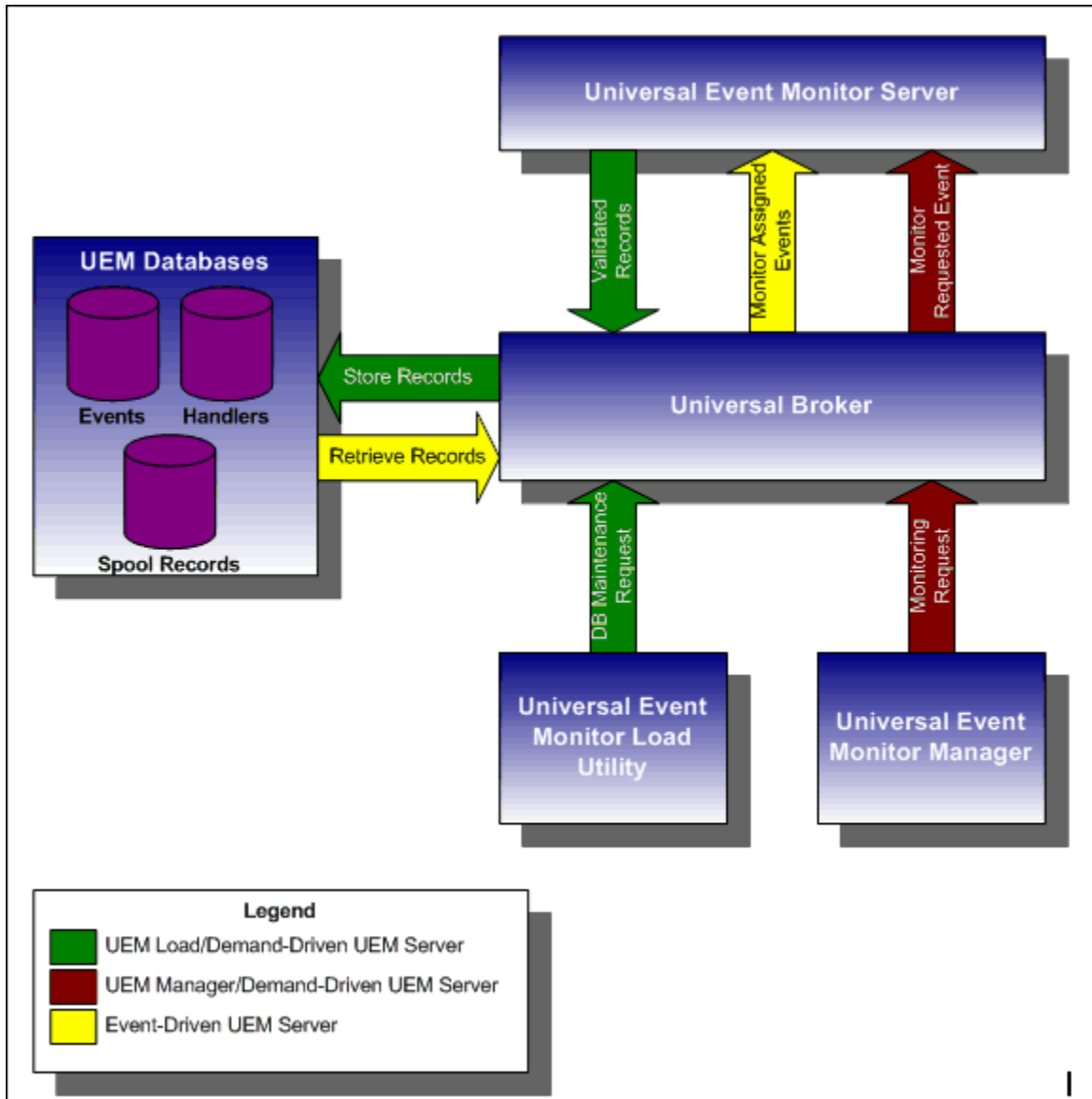


Figure 1.1  High-Level Interaction of UEM Components

# 1.3  Storing Event Definitions and Event Handlers

Event definitions and event handlers can be stored in separate BerkeleyDB database files. When an event definition or event handler record is added to its respective database, a unique identifier must be specified. Whenever UEM is required to monitor an event or execute an event handler, only this ID needs to be referenced in order for UEM to obtain the corresponding event definition or event handler parameters.

UEMLoad initiates all UEM-related database requests. UEMLoad is a command line application that can be used to:

- Add, update, and delete event definition and/or event handlers from their respective databases
- List the entire contents of the event definition and/or event handler databases
- List the parameters of a single event definition and/or event handler
- Export the contents of the event definition and/or event handler databases to a file that can be used to re-initialize the database or populate a new database on another system.

When UEMLoad is started, it sends a request to a Universal Broker running on the local system to start a UEM Server process. Because a client application (that is, UEMLoad) initiates the request, the UEM Server that is started is a demand-driven Server.

UEMLoad forwards the database request to the UEM Server, which validates it and supplies default values for any required parameters (based upon the type of request) that were not specified from the UEMLoad command line. When a set of complete, valid parameters is available, the UEM Server sends a request to the Universal Broker, which is responsible for actually performing the requested database operation.

Universal Broker reports the success or failure of all database maintenance requests (add, update, delete) to the UEM Server. The UEM Server then passes any errors back to UEMLoad.

For a database query request (list, export), Universal Broker will return the contents of each requested event definition or event handler record to the UEM Server, which then is responsible for forwarding the records to the UEMLoad.

Figure 1.2, below, illustrates the interaction of the Universal Broker and the Universal Event Monitor Server components involved during the execution of UEMLoad.
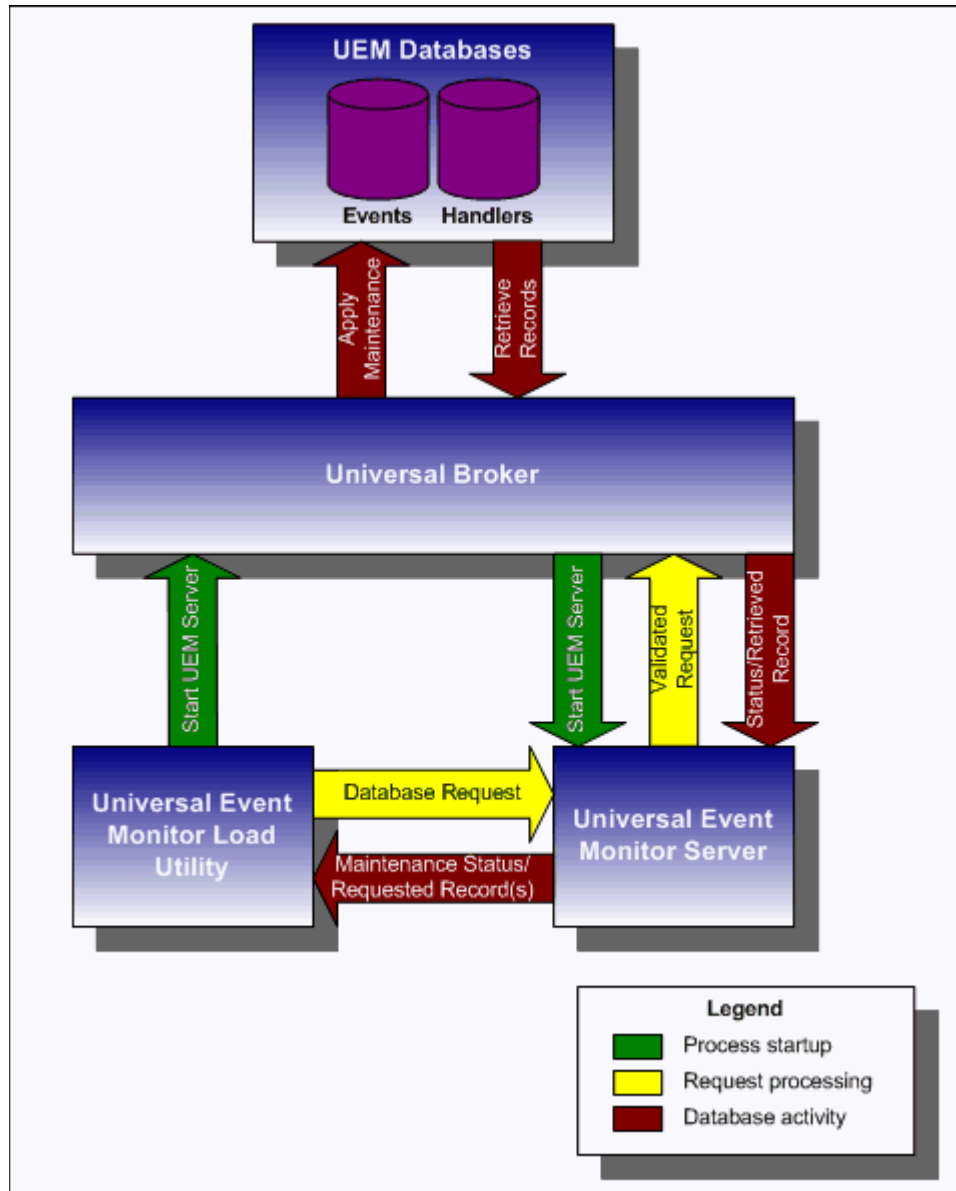


Figure 1.2  UEMLoad Utility Overview

# 1.4  Monitoring a Single Event

A single event can be monitored using the UEM Manager. The UEM Manager provides a command line interface from which all parameters required to define an event and its associated event handlers can be specified. In addition, the ID of a stored event definition or event handler can be used as an alternative to specifying all parameters explicitly.

When a UEM Manager is started, it sends a request to the specified local or remote Universal Broker to start a UEM Server. Because the request to start the UEM Server comes from a client application (that is, UEM Manager), it is a *demand-driven* Server that is started.

The UEM Manager sends the monitoring request to the UEM Server. The UEM Server validates the request and supplies default values for any required parameters that were not specified from the command line.

The UEM Manager command line provides for the assignment of an event handler to execute whenever the UEM Server sets the state of an event occurrence or state of the event itself. The UEM Server then is responsible for executing the assigned event handlers which are appropriate for the state change.

The UEM Server will monitor the event until either of the following conditions is satisfied:

- Required number of expected event occurrences has been detected
- Inactive date and time specified for the event definition elapses.

When either of these occurs, the event becomes inactive and the UEM Server stops monitoring it. The UEM Server then ends after informing the UEM Manager of the result of the monitoring request. The UEM Manager will set its exit code based on this information. This is the default behavior.

However, if an option was set in the UEM Manager instructing it to not wait on the UEM Server, the UEM Manager will end as soon as the UEM Server acknowledges its receipt of a valid monitoring request.

Figure 1.3, below, illustrates the interaction of the Universal Broker and the Universal Event Monitor components involved when a UEM Manager is executed.
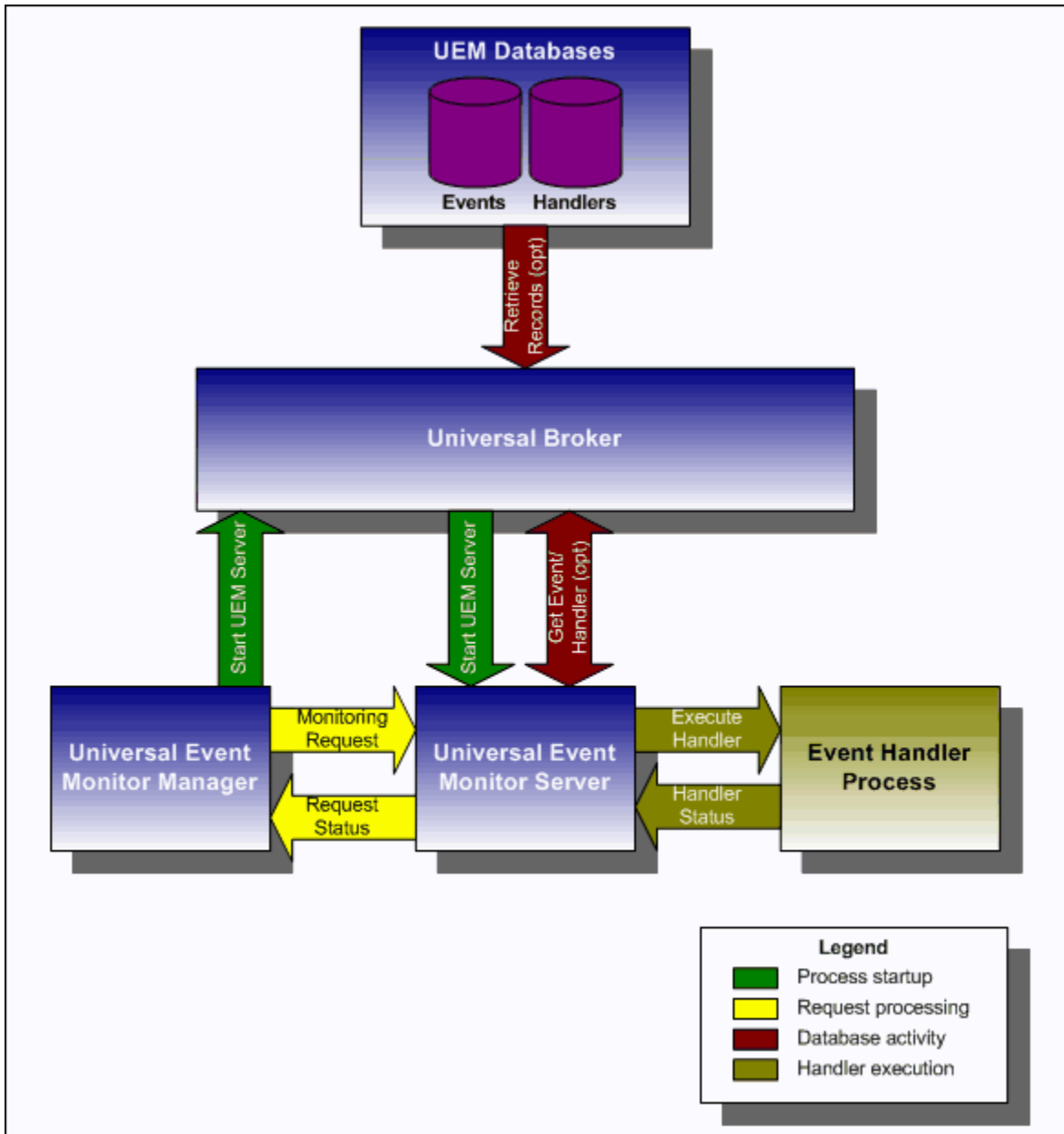


Figure 1.3  UEM Manager Overview

# 1.5  Monitoring Multiple Events

An *event-driven* Universal Event Monitor Server can be used to monitor multiple events at the same time. An event-driven UEM Server uses the records stored in the event definition database file to identify the events it is responsible for monitoring.

An event-driven UEM Server can be executed automatically during start-up of a Universal Broker. While it requires no interaction from a UEM client application, however, an event-driven UEM Server can be started at any time using Universal Control.

Unless it is stopped manually (using Universal Control), the event-driven UEM Server will continue to run as long as the Broker remains active. When the Broker stops, it will send a stop request to the UEM Server, instructing it to shut itself down.

When an event-driven UEM Server starts, it sends a request to the Broker asking for all of the event definitions residing in the event definition database that are assigned to that event-driven UEM Server. (This assignment was made when the event definition record was added to the database with UEMLoad.) The Server checks the active and inactive dates and times of the event definitions that it receives. It then begins monitoring the active events.

Each event definition provides for the assignment of an event handler to execute when an event occurrence is triggered or rejected. The assignment of an event handler to execute when an event expires also is made within the event definition. The UEM Server is responsible for executing appropriate event handlers based upon the states it sets for detected event occurrences and/or the event themselves.

Figure 1.4, below, illustrates the interaction of the Universal Broker and an event-driven UEM Server.
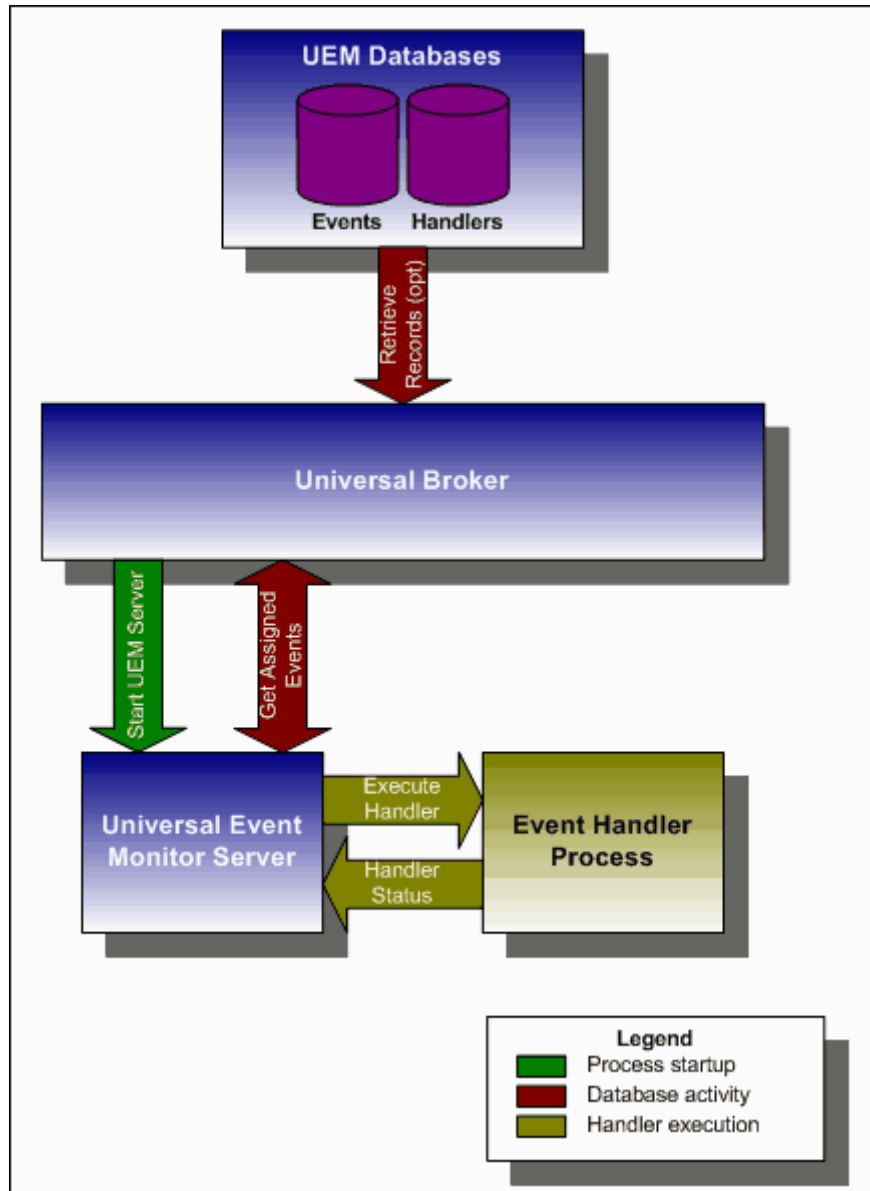


Figure 1.4  UEM Server Overview

# Chapter 2
# Features

## 2.1  Overview

A common user experience that employs standard conventions and a consistent feature set is a primary goal of all Stonebranch Inc. applications.

To that end, the methods, conventions, and features found with the Universal Event Monitor (UEM) application components are similar to those that can be found with other Universal Products.

This chapter provides information on Universal Event Monitor features that apply to all operating systems.

- Configuration
- Universal Configuration Manager
- Remote Configuration
- Universal Products Protocol
- Universal Products Databases
- Universal Access Control List
- Message and Audit Facilities

# 2.2  Configuration

Product configuration consists of specifying options that control product behavior and resource allocation.

- An example of configurable product behavior is whether or not data transferred over the network is compressed.
- An example of configurable resource allocation is the directory location in which the product creates its log files.

Each option is comprised of a pre-defined parameter, which identifies the option, and one or more values. The format of the parameter depends on the method being used to specify the option.

Although there are many configurable product options, Universal Products, in general, are designed to require minimal configuration and administration. The default options will work very well in most environments. When local requirements do require a change in product configuration, there are multiple methods available to configure the products in order to meet your needs.

# 2.2.1  Configuration Methods

All Stonebranch Inc. Universal Products provide a consistent and flexible method of configuration. An operating system's native configuration methods, such as configuration files, are utilized in order to integrate with existing system management policies and procedures for the platform.

Depending on specific Universal Products, and the operating system on which it is being run, product configuration is performed by one or more methods.  These configuration methods, in their order of precedence, are:

1. Command Line
2. Command Line File
3. Environment Variables
4. Configuration File

This order of precedence means that a command option specified on the command line overrides the same option specified in a command file, which overrides the same option specified with an environment variable, which overrides the same option specified with a configuration file keyword.

For security reasons, not all options can be overridden. Only applications run from the command line can override stored configuration options. In the Universal Products suite, these applications include manager components such as Universal Command Manager and Universal Event Monitor Manager. Other command line applications include utilities such as Universal Copy, Universal Query, and the Universal Event Monitor Load utility.

Server components such as Universal Event Monitor Server and service applications like Universal Broker rely solely on stored configuration options.

## 2.2.2  Command Line

Command line options consist of an option name and an option value. Configuration options specified with command line parameters override all corresponding options that are set with the other available configuration methods. If an option is specified more than once on the command line, the last instance of the option specified is used.

Configuration options overridden from the command line are only effective for a single instance of a given program. This allows the behavior of an application to be tailored to the specific needs of each program invocation.

Depending on the option specified, option values may or may not be case-sensitive. For example, an option whose only acceptable values are *yes* or *no* would not be case-sensitive. The option's value can be specified as *YES*, *yEs*, or *yes*. However, if the value specifies a directory name or file name, it may be case-sensitive, depending on whether or not the file system on the target platform is case-sensitive.

Other platform-specific differences and syntax rules are noted below.

**z/OS**

The z/OS command line options are specified with the PARM keyword of the JCL EXEC statement or with the SYSIN DD statement. The SYSIN DD statement offers more flexibility than the PARM keyword because it does not impose a limitation on length and it allows options to be specified on multiple lines. However, because all columns of the records are used as input, the data set or inline data allocated to the SYSIN DD statement cannot contain line numbers in the last 8 columns.

Regardless of the method used, the syntax is the same. Options are prefixed with a dash character (-). For many options, there are two different forms in which they can be specified: a short form and a long form. The long form, which is available for all options, consists of two or more *case-insensitive* characters. An optional short form consists of a single *case-sensitive* character.

The option value and option name must be separated by at least one space.

The following is an example of command line options specified using the PARM keyword:

```
Long form:
PARM='-LEVEL INFO -LOGIN YES'
Short form:
PARM='-l INFO -G yes'
```

**UNIX and Windows**

On Unix and Windows platforms, command line options are prefixed with a dash ( **-** ) character. On Windows, the slash ( **/** ) character can be used in place of the dash. For many options, there are two different forms in which they can be specified: a short form and a long form. The long form, which is available for all options, consists of two or more *case-insensitive* characters. An optional short form consists of a single *case-sensitive* character.

The option value and option name must be separated by at least one space or tab character.

The following demonstrates how to specify command line options on either platform:

```
Long form:
-level info -login yes
-LEVEL info -LoGiN YES
Short form:
-l info -G yes
```

## 2.2.3 Command Line File

Command line options that are used frequently and that are capable of being shared across Universal Products (for example, the user ID and password of a user account on the target system) can be stored in a command file and referenced as needed.

Options specified from a command file override environment variables and stored configuration options, and are overridden by options specified from the command line.

The individual options can be specified on one or more lines. While the option value must start on the same line as the command option, the value may be split across lines, provided it is enclosed in double quotes. No line continuation character is required to split option values across lines, but any spaces or tabs at the end and beginning of each line is preserved. Blank lines are ignored. Lines starting with the hash (**#**) character are ignored and can be used for comments.

### Examples of Valid Command Syntax

Some examples that demonstrate valid command file syntax are shown below. In each case, the values read by the application are the same.

**Command option and option value on one line**
```
-userid uid -pwd a_really_long_password
```

**Command options on multiple lines**
```
-userid uid
-pwd a_really_long_password
```

**Command options on multiple lines**
**(value for the `-pwd` option is split across lines)**
```
-userid uid
# The actual password value is: a_really_long_password.  Double
# quotes (") are required.
-pwd "a_really_
long_
password"
```

## Examples of Invalid Command Syntax

Some examples that demonstrate invalid command file syntax are shown below.

**Command option and option value on separate lines:**
```
# The start of the option value ('a_really_long_password') for
# the -pwd command option does not start on the same line as the
# option.

-userid uid -pwd
a_really_long_password
```

**Command value split, but not enclosed in double quotes:**
```
# The start of the option value ('a_really_long_password') for
# the -pwd command option starts on the same line as the option,
# but it is not enclosed in double quotes.  The application
# interprets 'long_' and 'password' as the beginning of new,
# invalid command options.

-userid uid
-pwd a_really_
long_
password
```

If it is necessary to secure the contents of a command file, the file can be encrypted using the Universal Encrypt utility. Please note that if the file contains sensitive material, the platform's native file and user security facilities should be also used to provide another level of protection against unintentional or unauthorized access to the file.

Unless otherwise noted, the **–file** and **–encryptedfile** command line options are provided by most Universal command line applications to specify a command file or encrypted command file, respectively.

# 2.2.4 Environment Variables

Environment variables, like command line options, allow options to be specified for one instance of a program execution. Each time that you execute a program, the behavior can be tailored to meet the specific needs for that execution with environment variables. Environment variables override stored configuration settings; they, in turn, are overridden by command file and command line options.

All environment variables used by Universal Products are upper case and begin with a product identifier consisting of three or four characters. Specific environment variable names can be found later in this document. Values are case-sensitive.

Methods for setting environment variables on different platforms are described below.

### z/OS

Environment variables are specified using the PARM keyword of the JCL EXEC. Environment variables are part of the IBM Language Environment (LE) and as such are specified as LE runtime options. The PARM value is divided into LE options and application options by a slash character (/). Options to the left of the slash are LE options and options to the right are application options.

The example below is another form of the examples shown above, and demonstrates how to set configuration options using an environment variable and a command line option. The message level is set for UEM using the **UEMLEVEL** environment variable while the **–login** option is set using normal command line syntax. Keep in mind that command line options may also be specified using the SYSIN DD statement (for more information, see Section 2.2.2 Command Line).

```
PARM='ENVAR("UEMLEVEL=INFO")/-login yes'
```

### UNIX

Environment variables in Unix are part of the shell environment, and can only be set using shell commands and conventions. To make it accessible to the program being executed, the environment variable must first be exported with the **export** shell command.

The example below demonstrates how to set the message level option to a value of INFO using the UEMLEVEL environment variable. Because the environment variable is exported, its value will be available to the Universal Event Monitor program. This example will work in either the bourne, bash, or korn shells.

```
UEMLEVEL=INFO
export UEMLEVEL
```

### Windows

Environment variables in Windows are defined as part of the Windows console command environment. As such, console commands are used to set environment variables.

Example of setting an environment variable:

```
Set option UEMLEVEL to a value of INFO:
SET UEMLEVEL=INFO
```

# 2.2.5  Configuration File

Configuration files are used to specify system-wide configuration values. They are last in the precedence order for specifying configuration options (see Section 2.2.1 Configuration Methods).

(For most Universal Products, some options can be specified only in a configuration file, while other options can be overridden by individual command executions. The Stonebranch, Inc. documentation for each product identifies these options.)

If an option is specified more than once in a configuration file, the last option specified is used.

All configuration files on a system are managed by the local Universal Broker. The Universal Broker serves the configuration data to other Universal Products running on the local system. The one exception is Universal Enterprise Controller (UEC). UEC directly reads its own configuration files.

The Universal Broker reads the configuration files when it first starts or when it receives a REFRESH command from Universal Control or Universal Enterprise Controller. Any changes made to a configuration file are not in effect until the Broker is recycled or receives a REFRESH command.

Universal Product components do not read the configuration files themselves. When a component starts, it first registers with the locally running Universal Broker. As part of the registration process, the Broker returns the components configuration data.

When the Universal Broker is operating in managed mode, the configuration information for the various Universal Products is "locked down" and can be modified or viewed only via the Universal Management Console (see Section 2.3.2 Managed Mode).

---

**z/OS**

Configuration files are members of a PDSE. The data set record format is fixed or fixed block with a record length of 80. No line numbers can exist in columns 72-80. All 80 columns are processed as data.

All configuration files are installed in the **UNVCONF** library.

See Section  2.2.6 Configuration File Syntax for the configuration file syntax.

---

**UNIX**

Universal Products components search for their configuration files in a fixed list of directories. The products will use the first configuration file it finds in its search.

The order in which the directories are searched is as follows:

1. /etc/opt/universal
2. /etc/universal (default installation location)
3. /etc
4. /usr/etc/universal
5. /usr/etc

If an option is specified more then once in a configuration file, the last option specified is used.

See Section 2.2.6 Configuration File Syntax for the configuration file syntax.

---

**Windows**

On Windows, configuration files are stored in the **%ALLUSERSPROFILE%\Application Data\Universal\conf** directory, where **%ALLUSERSPROFILE%** is a system environment variable that represents the location of the well-known "All Users" folder. This complete path to this directory is typically **C:\Documents and Settings\All Users**.

On Windows Vista and Windows Server 2008, **%ALLUSERSPROFILE%** resolves by default to **C:\Program Data**, making the default path to configuration files **C:\Program Data\Universal\conf**.

Although configuration files can be edited with any text editor (for example, Notepad), the Universal Configuration Manager application, accessible via the Control Panel, is the recommended way to set configuration options (see Section 2.4 Universal Configuration Manager).

Universal Configuration Manager provides a graphical interface and context-sensitive help, and helps protect the integrity of the configuration file by validating all changes to configuration option values. However, should a situation arise in which a configuration file must be edited directly, you must follow the syntax conventions below to prevent corruption of that file.

# 2.2.6 Configuration File Syntax

Configuration files are text files that can be edited with any available text editor.

The following rules apply for configuration file syntax:

- Options are specified in a keyword / value format.
- Keywords can start in any column.
- Keywords must be separated from values by at least one space or tab character.
- Keywords are not case sensitive.
- Keywords cannot contain spaces or tabs.
- Values can contain spaces and tabs, but if they do, they must be enclosed in single ( **'** ) or double ( **"** ) quotation marks. Repeat the enclosing characters to include them as part of the value.
- Values case sensitivity depends on the value being specified. For example:
    - Directory and file names are case sensitive.
    - Pre-defined values (such as **yes** and **no**) are not case sensitive.
- Each keyword / value pair must be on one line.
- Characters after the value are ignored.
- Newline characters are not permitted in a value.
- Values can be continued from one line to the next either by ending the line with a:
    - Plus ( **+** ) character, to remove all intervening spaces.
    - Minus ( **-** ) character, to preserve all intervening spaces between the end of the line being continued and the beginning of the continuing line.

    Ensure that the line continuation character is the last character on a line.
- Comment lines start with a hash (**#**) character.
- Blank lines are ignored.

Note: If an option is specified more than once in a configuration file, the last option specified is used.

# 2.3  Remote Configuration

Universal Products can be configured remotely by Universal Enterprise Controller using the Universal Management Console (UMC) client application. UMC instructs the Universal Broker of a remote Universal Agent to modify the configurations of the Universal Products components managed by that Broker.

Universal Broker supports remote configuration in either of two modes:

1.  Unmanaged Mode
2.  Managed Mode

## 2.3.1  Unmanaged Mode

Unmanaged mode is the default mode of operations for Universal Broker. It allows a Universal Broker – and the Universal Products managed by that Universal Broker – to be configured either:

•   Locally, by editing configuration files.
•   Remotely, via UMC.

The system administrator for the machine on which a Universal Agent resides can use any text editor to modify the configuration files of the various local Universal Products.

Via UMC, selected users can modify all configurations of any Universal Agent, including the local Universal Agent. UMC sends the modified data to the Universal Broker of that agent, which Universal Broker then uses to update the appropriate configuration files.

If UMC sends modifications for a Universal Broker configuration, Universal Broker validates the modified data before it accepts it. If the data fails validation, Universal Broker does not update its configuration file.

If UMC sends modification to the configuration of any other Universal Products component, the Universal Broker updates the appropriate configuration file. The component will use this new configuration at its next invocation.

Note:   If errors or invalid configuration values are updated via UMC for a component other than Universal Broker, the component may not run successfully until the configuration has be corrected.

# 2.3.2  Managed Mode

When a Universal Broker is operating in managed mode, the configuration information for all Universal Products components managed by that Universal Broker is "locked down." Universal Broker stores the information in a database file located within its specified spool directory. The information can be modified only via Universal Management Console (UMC).

From this point on, Universal Broker uses the database file – not the configuration files – to access configuration information. Any configuration changes made to the components – via UMC – are placed in the database file. Therefore, as long as Universal Broker stays in managed mode, the configuration files may no longer contain current or valid configuration information.

If managed mode is de-selected for the Universal Broker, it reads the database file where it stored the configuration information. Universal Broker uses this information to create and/or update configuration files for the components.

*   If a configuration file exists in the configuration directory, it is overwritten.
*   If a configuration file does not exist, it is created.

Note:   Because of remote configuration and the desire to be able to "lock down" all product configurations, Universal Broker – and all Universal Products servers – no longer support the command line and environmental variables methods of specifying configuration options.

## Selecting Managed Mode

The managed mode of operations for Universal Broker is selected via the Universal Enterprise Controller Administration client application.

(See the Universal Enterprise Controller 3.2.0 Client Applications guide for specific information on how to select managed mode.)

Figure 2.1, below, illustrates remote configuration for one Universal Agent in managed mode and one Universal Agent in unmanaged mode.
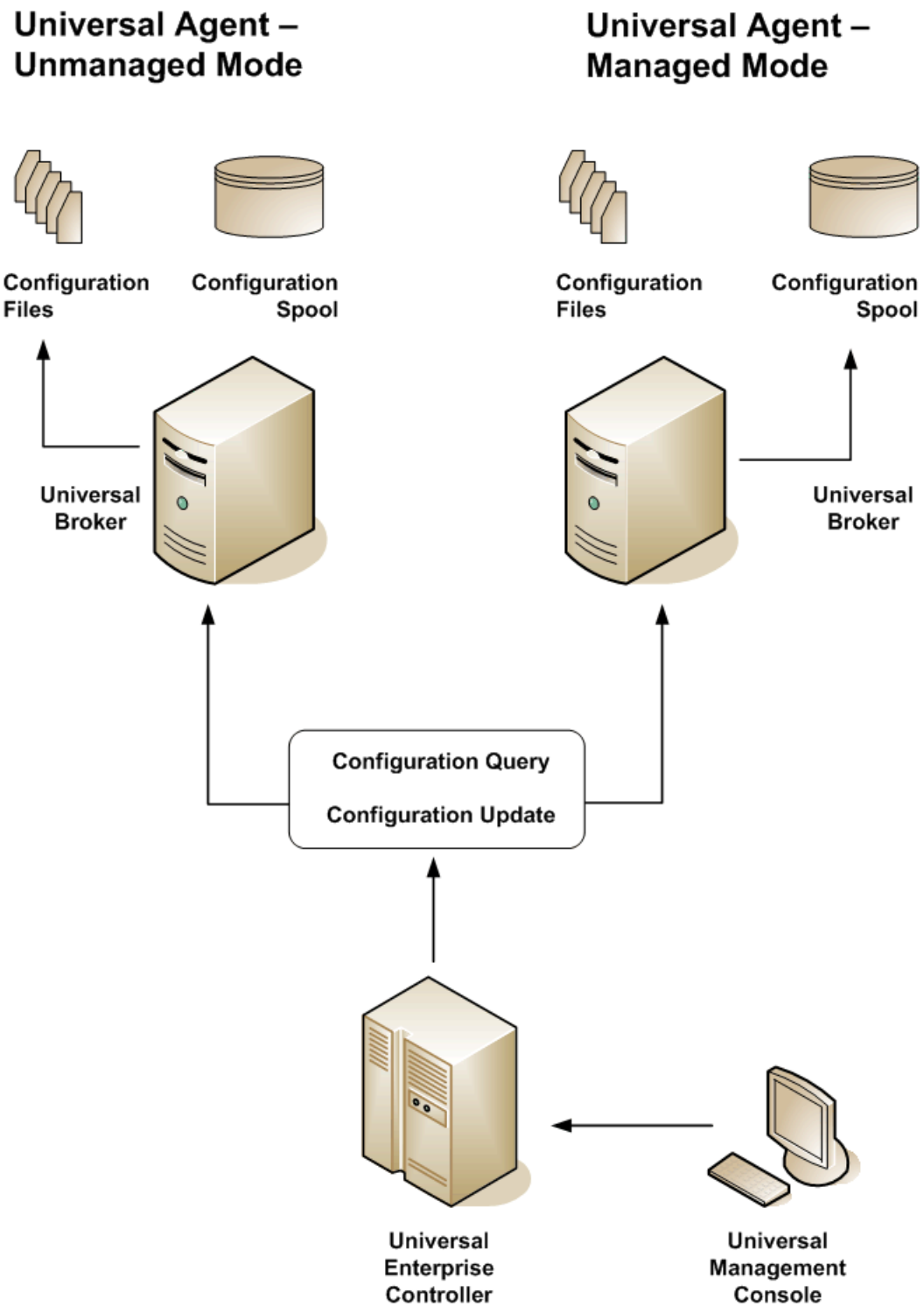


Figure 2.1  Remote Configuration - Unmanaged and Managed Modes of Operation

## 2.3.3  Universal Broker Start-up

At Universal Broker start-up, in both managed and unmanaged modes, the Universal Broker configuration file always is read.

### Unmanaged Mode

At Universal Broker start-up in unmanaged mode, Universal Broker reads the configuration files of all Universal Products components into its memory. The Universal Broker configuration file is used to define the Universal Broker configuration, just as all configuration files are used in unmanaged mode. Universal Broker updates its memory from the configuration files whenever Universal Control issues a REFRESH request.

### Managed Mode

At Universal Broker start-up in managed mode, the Universal Broker configuration file points Universal Broker to the location of the configuration spool file, from which the Broker retrieves configuration information for all Universal Products. Universal Broker updates its memory from the configuration spool file and, automatically, after changes are made via UMC.

If more configuration information than needed is included in the Universal Broker configuration file at Universal Broker start-up, Universal Broker will update its running configuration with the information that it retrieved from the spool file. The configuration file that was used at start-up is made obsolete.

# 2.4  Universal Configuration Manager

The Universal Configuration Manager is a Universal Products graphical user interface application that enables you to configure all of the Universal Products that have been installed on a Windows operating system.

It is the recommended method of specifying configuration data that will not change with each command invocation. Universal Configuration Manager helps protect the integrity of the configuration file by validating all changes to configuration option values.

## 2.4.1  Availability

Universal Configuration Manager is installed automatically on the Windows operating system as part of every Universal Products for Windows installation.

It is available to all user accounts in the Windows Administrator group.

**Windows Vista**

When opening the Universal Configuration Manager for the first time on Windows Vista, two new operating system features, the Program Compatibility Assistant (PCA) and User Account Control (UAC), may affect its behavior.

With these two features enabled, the expected Universal Configuration Manager behavior is as follows:

1.   Universal Configuration Manager may issue the following error:
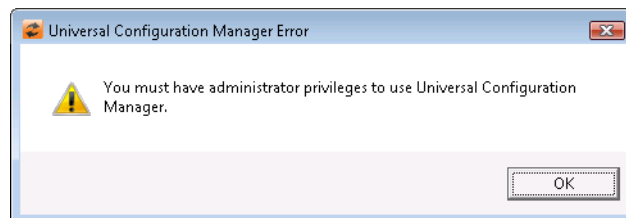
Figure 2.2  Universal Configuration Manager Error dialog - Windows Vista

2.   Click **OK** to dismiss the error message.

The Windows Vista Program Compatibility Assistant (PCA) displays the following dialog:
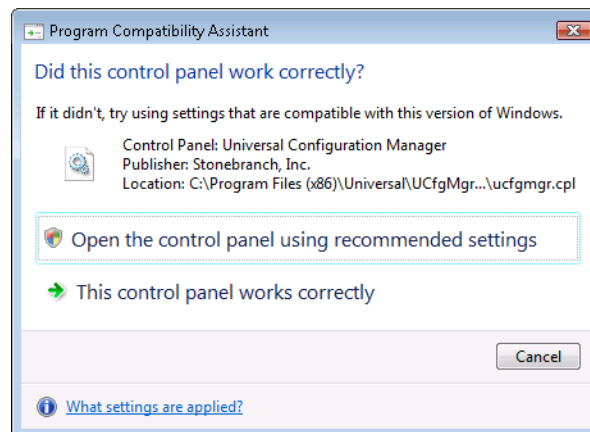
Figure 2.3  Windows Vista - Program Compatibility Assistant

3.  To continue, select **Open the control panel using recommended settings**.  This instructs the PCA to "shim" (Microsoft term) the Configuration Manager, establishing it as an application that requires elevated privileges.

    Windows Vista User Account Control (UAC) then displays a prompt seeking permission to elevate the logged-in account's access token.

4.  Select **Continue** to give the account full administrative privileges.

    Subsequent attempts to open Universal Configuration Manager should result only in the UAC prompt.

# 2.4.2 Accessing the Universal Configuration Manager

To access the Universal Configuration Manager:

1. Click the `Start` icon at the lower left corner of your Windows operating system screen to display the Start menu.
2. Click (Settings/) Control Panel on the Start menu to display the Control Panel screen.
3. Select the Universal Configuration Manager icon to display the Universal Configuration Manager screen (see Figure 2.4).

**Windows XP, Windows Vista, Windows Server 2008**

Newer versions of Windows support a Control Panel view that places applet icons within categories. This "category view" may affect the location of the Universal Configuration Manager icon.

For example, the Windows XP Category View places the Universal Configuration Manager icon under the `Other Control Panel Options` link.  Windows Vista and Windows Server 2008 place the icon within the `Additional Options` category.

If you have trouble locating the Universal Configuration Manager icon, simply switch to the Classic View to display all Control Panel icons at the same time.

**64-bit Windows Editions**

The Windows Control Panel places icons for all 32-bit applets under the `View x86 Control Panel Icons` (or, on newer versions, the `View 32-bit Control Panel Icons`) category, even when the Classic View is enabled.

When using the Category View, look for the 32-bit Control Panel applet icons in the `Additional Options` category.
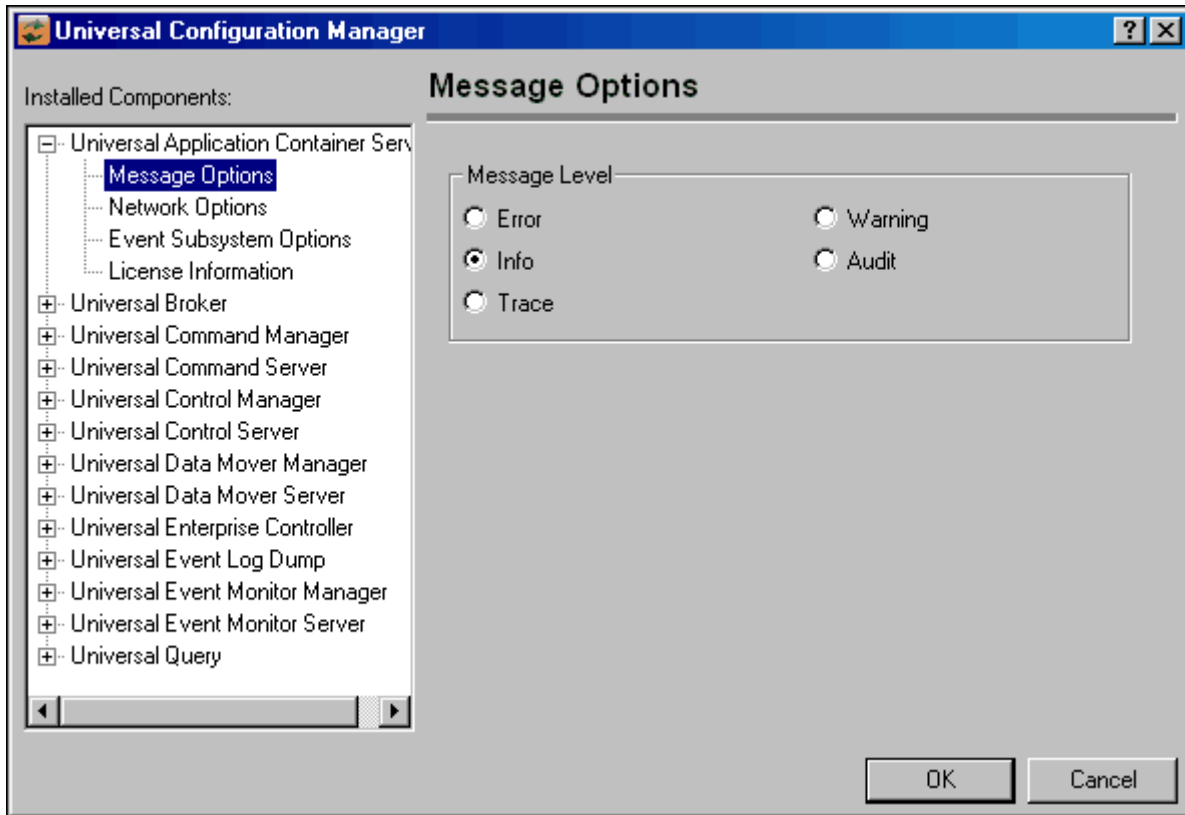
Figure 2.4  Universal Configuration Manager

Each Universal Configuration Manager screen contains two sections:

1.  Left side of the screen displays the Installed Components tree, which lists:
    *   Universal Products components currently installed on your system.
    *   Property pages available for each component (as selected), which include one or more of the following:
        *   Configuration options
        *   Access control lists
        *   Licensing information
        *   Other component-specific information
2.  Right side of the screen displays information for the selected component / page.

(By default, Universal Configuration Manager displays the first property page of the first component in the Installed Components tree.)

## 2.4.3  Navigating through Universal Configuration Manager

To display general information about a component, click the component name in the Installed Components list.

To display the list of property pages for a component, click the **+** icon next to the component name in the Installed Components list.

To display a property page, click the name of that page in the Installed Components list.

If a property page has one or more of its own pages, a **+** icon displays next to the name of that property page in the Installed Components list. Click that **+** icon to display a list of those pages.

In Figure 2.4, for example:

- List of property pages is displayed for Universal Broker.
- Message Options property page has been selected, and information for that property is displayed on the right side of the page.
- No **+** icons next to any of the property pages indicates that they do not have one or more of their own property pages.

## 2.4.4  Modifying / Entering Data

On the property pages, modify / enter data by clicking radio buttons, selecting from drop-down lists, and/or typing in data entry fields.

Some property pages provide panels that you must click in order to:

- Modify or adjust the displayed information.
- Display additional, modifiable information.

Note:   You do not have to click the **OK** button after every modification or entry, or on every property page on which you have modified and/or entered data. Clicking **OK** just once, on any page, will save the modifications and entries made on all pages – and will exit Universal Configuration Manager (see Section 2.4.5 Saving Data.)

### Rules for Modifying / Entering Data

The following rules apply for the modification and entry of data:

- Quotation marks are not required for configuration values that contain spaces.
- Edit controls (used to input free-form text values) handle conversion of any case-sensitive configuration values. Except where specifically noted, values entered in all other edit controls are case insensitive.

## 2.4.5  Saving Data

To save all of the modifications / entries made on all of the property pages, click the `OK` button at the bottom of any property page. The information is saved in the configuration files, and Universal Broker is automatically refreshed.

Clicking the `OK` button also exits the Universal Configuration Manager. (If you click `OK` after every modification, you will have to re-access Universal Configuration Manager.)

To exit Universal Configuration Manager without saving any of the modifications / entries made on all property pages, click the `Cancel` button.

## 2.4.6  Accessing Help Information

Universal Configuration Manager provides context-sensitive help information for the fields and panels on every Universal Products component options screen.

To access Help:

1. Click the question mark ( **?** ) icon at the top right of the screen.
2. Move the cursor (now accompanied by the **?**) to the field or panel for which you want help.
3. Click the field or panel to display Help text.
4. To remove the displayed Help text, click anywhere on the screen.

**Windows Vista, Windows Server 2008**

The Universal Configuration Manager's context-sensitive help is a WinHelp file, which Windows Vista and Windows Server 2008 does not support.

Microsoft offers the 32-bit WinHelp engine as a separate download from its website. If you require access to the Universal Configuration Manager's context-sensitive help, simply download and install the WinHelp engine.

# 2.4.7  UEM Installed Components

## UEM Manager

Figure 2.5 illustrates the Universal Configuration Manager screen for the Universal Event Monitor Manager.

The Installed Components list identifies all of the UEM Manager property pages.

The text describes the selected component, Universal Event Monitor Manager.
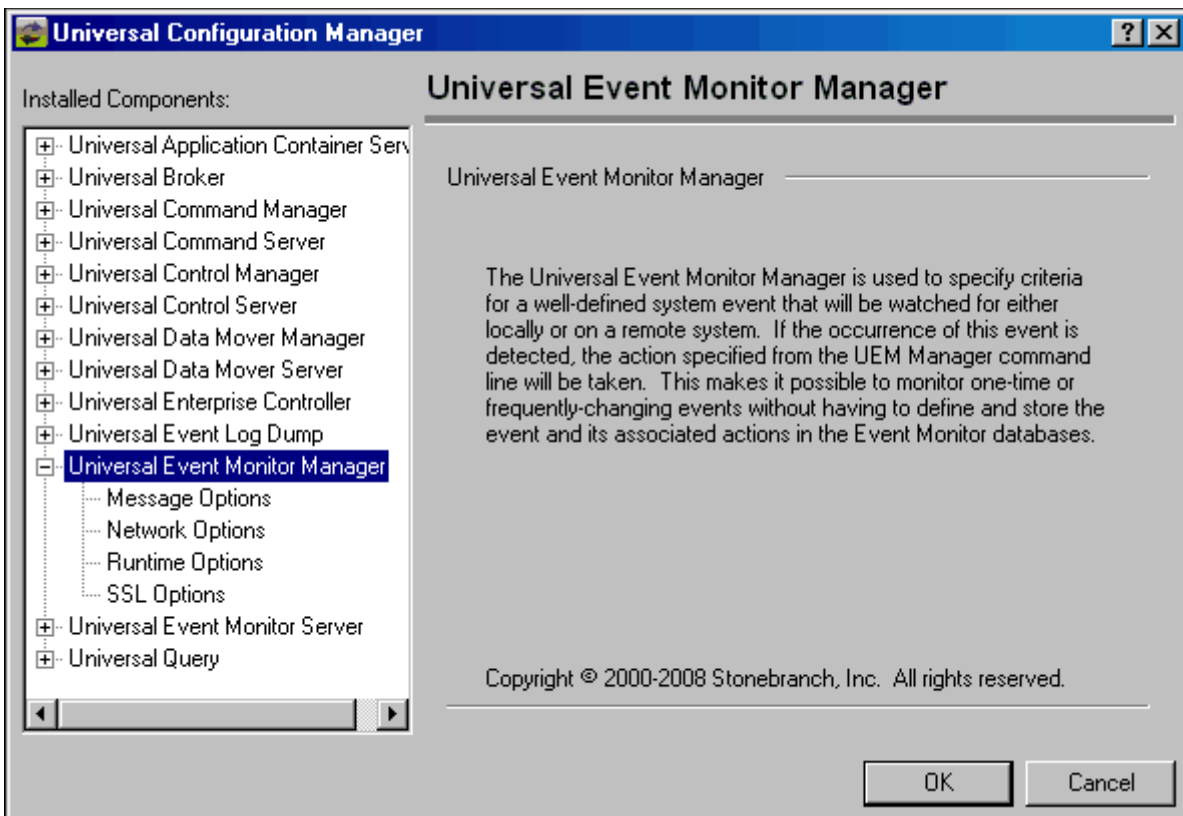


Figure 2.5  Universal Configuration Manager - UEM Manager

## UEM Server

Figure 2.6 illustrates the Universal Configuration Manager screen for the Universal Event Monitor Server.

The Installed Components list identifies all of the UEM Server property pages.

The text describes the selected component, Universal Event Monitor Server.
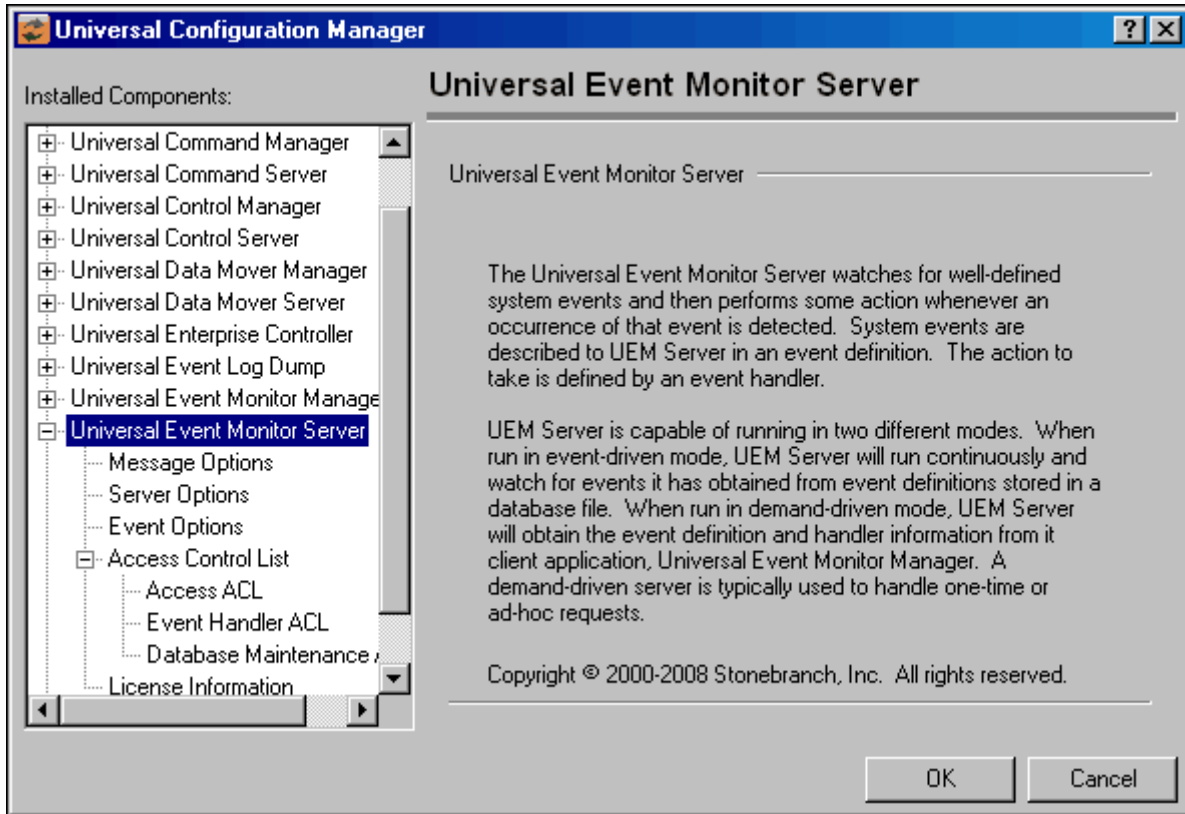


Figure 2.6  Universal Configuration Manager - UEM Server

# 2.5  Universal Products Protocol

The components that comprise distributed applications such as UEM require the ability to communicate with each other over data networks. All Universal Products components, including those distributed with UEM, use a common application layer protocol to exchange data messages.

The following sections describe the security, extensibility, and configurability of the Universal Products application layer protocol.

# 2.5.1  Security

The security features used in the Universal Products application protocol are applied to all messages exchanged between Universal Products components. They cannot be turned off.

This section begins with a general discussion of security issues, specifically data privacy and integrity, and then describes the means by which these issues are addressed by Universal Products components.

## Data Privacy and Integrity

People with sufficient technical knowledge and access to network resources can watch or capture data transmitting across the network. The captured data is then at their mercy.

Sensitive data sent over the network must be encrypted in such a manner that unauthorized persons cannot determine the original content of the data regardless of their level of expertise, access to network resources, amount of data captured, and the amount of time they have. The intended recipient is the only party that should be able to read the data.

Encryption algorithms are used to encrypt data into an unreadable format. The encryption process is computationally expensive. There are a variety of encryption algorithms, some of which perform better than others. Some algorithms offer a higher level of security than others. Typically, the higher level of security requires more computational resources.

Data integrity must be protected from errors in transmission and from malicious users. As data is transmitted over the network, it passes through media and hardware of unknown quality that may erroneously change bits of data without warning. Even data that is encrypted is susceptible to being changed by a malicious party while it is transmitted over the network. The changed data may or may not be detected by the recipient, depending on what changed and how it is processed. The data may be accepted as valid even though it is, in fact, corrupted. Data integrity checks must therefore be done to ensure that what is received matches what was sent.

Message digest algorithms are used to produce a Message Authentication Code (MAC) that uniquely identifies a block of data. The sender computes a MAC for the data being sent based on a shared secret key the sender and receiver hold. The sender sends the data and the MAC to the receiver. The receiver computes a new MAC for the received data based on the shared secret key. If the two MACs are the same, data integrity is confirmed and the data is accepted. Otherwise, the data is considered modified and is rejected.

To address these issues of data privacy and data integrity, all Universal Products components, starting with version 3.x, use the Secure Socket Layer (SSL) protocol. Versions of Universal Products components prior to version 3.x used a proprietary protocol to handle data privacy and integrity. To ensure backward compatibility, this protocol is still supported by version 3.x components and is discussed in Section 2.3.1.3 Universal Products Protocol.

## Secure Socket Layer Protocol

The Secure Socket Layer (SSL) protocol is a widely used and accepted network protocol for distributed software applications that are required to address all aspects of secure data transfer on private and public networks. SSL provides the highest level of security available.

Universal Products implement the SSL protocol using the OpenSSL library. The most recent SSL standard is version 3. A subsequent version was produced changing the name to Transport Layer Security version 1 (TLSv1). TLSv1 is the actual protocol used by Universal Products. TLSv1 is more commonly referred to simply as SSL. The term SSL is used throughout the rest of this document to mean TLSv1, unless otherwise noted.

The SSL standard defines a set of encryption and message digest algorithms referred to as cipher suites that ensure data privacy and data integrity. Cipher suites pair encryption algorithms with appropriate message digest algorithms. The two algorithms cannot be specified individually.

Universal Products provides support for a subset of the available SSL cipher suites. The cipher suite name is formatted as an encryption algorithm abbreviation followed by the message digest algorithm abbreviation.

Table 2.1, below, lists all supported cipher suites.

| Cipher Suite Name | Description |
|---|---|
| RC4-SHA | 128-bit RC4 encryption with SHA-1 message digest |
| RC4-MD5 | 128-bit RC4 encryption with MD5 message digest |
| AES256-SHA | 256-bit AES encryption with SHA-1 message digest |
| AES128-SHA | 128-bit AES encryption with SHA-1 message digest |
| DES-CBC3-SHA | 128-bit Triple-DES encryption with SHA-1 message digest |
| DES-CBC-SHA | 128-bit DES encryption with SHA-1 message digest |
| NULL-SHA | No encryption with SHA-1 message digest |
| NULL-MD5 | No encryption with MD5 message digest |

Table 2.1  Supported SSL Cipher Suites

### Selecting an SSL Cipher Suite

When two Universal Products components (for example, a UEM Manager and a UEM Server) first connect, they perform an SSL handshake that negotiates the cipher suite to use for the session. The UEM Manager presents a list of cipher suites (in descending order of preference) that it would like to use. This is compared against a list of ciphers that the UEM Server supports. The first cipher suite in common is the one used for the session.

Why is a list of cipher suites helpful? A distributed software solution may cross many organizational and application boundaries, each with its own security requirements. Instead of having to choose one cipher suite for all distributed components, the software components can be configured with their own list of acceptable cipher suites based on their local security requirements. When a high level of security is required, the higher CPU consuming cipher suite is justified. When lower level of security is acceptable, a lower CPU consuming cipher suite can be used. As long as the UEM Manager has both cipher suites in its list, it can negotiate either cipher suite with servers of different security levels.

## Universal Products Protocol

The Universal Products protocol (UNVv2) is a proprietary protocol that securely and efficiently transports data across data networks. UNVv2 is used in Universal Products prior to version 3.x and will continue to be provided for backward compatibility in future versions.

UNVv2 ensures data privacy through the use of 128-bit RC4 encryption. Encryption keys are generated using a proprietary key agreement algorithm. A new key is created for each network session.

Data integrity is implemented using message digest algorithms that build 128-bit MD5 MACs. UNVv2 referred to data integrity as data authentication.

# 2.5.2 Extensibility

The message protocol used between Universal Products components is extensible. New message fields can be added with each new release without introducing incompatibilities between different versions of product components. This is a very important feature for distributed systems since it is nearly impossible to upgrade hundreds of servers simultaneously.

New encryption and compression algorithms can be added in future releases without sacrificing compatibility with older releases. After a network connection is established between two Universal components, connection options, such as available encryption and compression algorithms, are negotiated. The result of this negotiation process is an agreed-upon algorithm that is implemented by both components.

# 2.5.3 Configurability

The Universal Products application protocol can be configured to meet the needs of any network environment. To this end, the UEM provides the network delay option.

## Network Delay

The network delay option provides the ability to fine tune Universal product's network protocol. When a data packet is sent over a TCP/IP network, the time it takes to reach the other end depends on many factors, such as, network congestion, network bandwidth, and the network media type. If the packet is lost before reaching the other end, the other end may wait indefinitely for the expected data. In order to prevent this situation, Universal Products time out waiting for a packet to arrive in a specified period of time. The delay option specifies this period of time.

The network delay option specifies the maximum acceptable delay in transmitting data between two programs. If a data transmission takes longer than the specified delay, the operation ends with a time-out error.

Note:   This configuration option only applies to transmission of data between two applications. Changing this value will not affect time-out errors that occur when network connections are first being established.

The default network delay value is 120 seconds. This value is reasonable for most networks and operational characteristics.

- If the value is too small, false network time-outs could occur.
- If the value is too large, programs will wait a long period of time before reporting a time-out problem.

# 2.6  Universal Products Databases

Universal Products utilize a set of databases to record information for Universal Products components managed by Universal Broker. There are some application-specific features in these components that rely upon these databases for their implementation.

Unless otherwise noted, the Universal Broker is the owner of all databases and is responsible for performing all direct database access. Any Universal Products application that requires access to a database must forward a request to the Universal Broker for processing.

## 2.6.1  Database Files

The database files that are relevant to Universal Event Monitor are:

- Component information database
- Event definition database
- Event handler database
- Event spool database

The component information database is used to record information about all Universal Products server components managed by Universal Broker. The other three databases are used exclusively for support of UEM.

The database files are local to each system. They reside in a directory that, if it does not already exist, is created during product installation. The component information database is opened during Universal Broker startup processing. Other databases are opened as needed. If an attempt is made to open a database that does not exist, the database file is created.

**UNIX**

The default database directory is `/var/opt/universal/spool`.

**Windows**

The default database directory is `C:\Program Files\Universal\spool`. Databases that are accessed exclusively by Universal Broker are stored in a subdirectory named `ubroker`.

Additional information that applies to all database files, including restrictions on location and space requirements, may be found in the Universal Products 3.2.0 Installation Guide.

## 2.6.2  Component Information Database

The component information database maintains a record of each Universal Products server component (including each UEM Server) started by Universal Broker. The information captured by the Universal Broker includes, but is not limited to, the component's process ID, start time, current state, and end time.

One important aspect of this database is its ability to record the current state of a Universal Products server component. Each time a component's state changes, it sends a notification to Universal Broker, which updates that component's record. This state serves as the basis for support of reconnect or restart functionality (that is, fault tolerance).

While fault tolerance is not currently implemented in UEM, a UEM Server component still notifies the Universal Broker whenever its state changes. The states reported by UEM are simply a subset of all available states used for fault tolerance support.

A list of active UEM Server components, along with their current state, can be viewed using the Universal Query utility. (See the Universal Products Utilities 3.2.0 User Guide for information on Universal Query.)

When a Universal Products server process finishes executing and its component state indicates that it has completed, Universal Broker deletes that component's information from the database. A UEM Server component always is considered complete as soon as the process finishes. So, when the UEM Server process ends, its component record is no longer available.

The name of the component information database file is `bcomponent.db`.

**Windows**

Because this database is only accessed by Universal Broker, it resides (by default) in the `.\Universal\spool\ubroker` directory.

## 2.6.3  Event Definition Database

The event definition database is used to store event definitions that can be referenced by a unique ID and used by a UEM Server process.

The parameters in an event definition record describe a system event and establish the criteria used to test for the completion of an event occurrence. The event definition record may also contain one or more ID's of records in the event handler database that defines the way in which an event or an event occurrence is handled (that is, responded to).

Stored event definitions are required for an event-driven UEM Server. When an event-driven UEM Server starts, it queries the Broker for all event definition records that are assigned to it. If an event-driven UEM Server has no event definitions assigned to it, it will continue to execute, but will not do any actual event monitoring.

Stored event definitions may also be used by a demand-driven UEM Server, if the UEM Manager specifies the ID of an event definition record from its command options.

Event definition records are added and maintained with the Universal Event Monitor Load utility. A complete description of the UEMLoad utility can be found in Chapter 10 Universal Event Monitor Load Utility.

# 2.6.4 Event Handler Database

The event handler database is used to store event handlers that can be referenced by a unique ID and used by a UEM Server process.

The parameters in an event handler record describe the action to take in response to an event or an event occurrence that has satisfied certain conditions. The action can be a system command or a script for UEM Server to execute. User account information also can be specified in the event handler, making it possible for the process started by UEM to execute in the security context of that user.



Security is a primary concern within all Universal Products.

Whenever the user account information
stored in an event handler record includes a password,
that password is encrypted
using the Data Encryption Standard (DES) algorithm.

**Stoneman's Tip**

Stored event handlers are required for an event-driven UEM Server. An event-driven UEM Server obtains information for the event handlers it executes using information contained in those event definitions. The event definition record includes parameters that may be used to specify the event handler to execute when the event or an event occurrence satisfies certain conditions.

Stored event handlers also can be used by a demand-driven UEM Server, if the UEM Manager specifies the ID of an event handler record from its command options.

When a UEM Server needs to use a stored event handler record, it sends a request to the Universal Broker to retrieve the record using the ID specified in the event definition record or provided from the UEM Manager command line. The Universal Broker returns the event handler record to the UEM Server, which then executes the specified system command or script.

Event handler records are added and maintained with UEMLoad. A complete description of UEMLoad can be found in Chapter 10 Universal Event Monitor Load Utility.

## 2.6.5  Event Spool Database

The event spool database is used to track the progress of each event monitored by a UEM Server. A record is created either whenever an occurrence of an event is detected or when an event expires. The record is used to track the status of that occurrence all the way through the completion of any event handlers that may be executed for the occurrence.

Because there may be multiple occurrences detected for any single monitored event, each event may result in the creation of several spool records. If no occurrence of an event is detected, and the monitored event becomes inactive, a single spool record will be written showing that the event expired.

Universal Broker applies all updates to the event spool database. A UEM Server is responsible for sending the Universal Broker all relevant information, along with the required database operation (add, update, or delete).

Typically, any spool records created for an event are deleted when the Broker detects the completion of the UEM Server that monitored the event. However, when an event-driven UEM Server completes, any records that indicate work in progress (for example, tracking of an event occurrence, execution of an event handler) are retained for possible recovery when the event-driven Server is restarted. For additional information on recovery of event spool records, see Chapter 7 Universal Event Monitor Server.

| | |
|---|---|
| | An option can be set in the Universal Broker's configuration to prevent it from deleting any event spool records when the UEM Server component completes. Setting the `comp_info_retention` option to a value greater than 0 causes the event spool record to be preserved. |
| **Stoneman's Tip** | Because there is currently no database cleanup routine available, this option should be set only following a recommendation from, and with the assistance of, Stonebranch Inc. Customer Support. |

Feedback from a demand-driven UEM Server is returned to the UEM Manager that initiated the monitoring request. In this situation, event spool records are simply another means of following the progress of the event and any detected occurrences.

However, for an event-driven UEM Server that has no client, the records in the event spool database are the best way to monitor the status of the work performed by that UEM Server. Because an event-driven UEM Server typically is a long-running process, an adequate history of the UEM Server's activity can be obtained by viewing the spool records.

Currently, event spool records can only be viewed with the Universal Spool List utility (`uslist`). Information on using Universal Spool List to view event spool records can be found in Chapter 7 Universal Event Monitor Server. For information on all utilities provided with Universal Products, see the Universal Products Utilities 3.2.0 User Guide.

## 2.6.6  Controlling Database Access

Universal Broker is responsible primarily for providing access to the Universal Products databases. However, there are utilities provided, including the Universal Spool List (`uslist`) and Universal Spool Remove (`uslrm`), that can be used to access the databases directly. While these utilities should be used only following a recommendation from and with the assistance of Stonebranch, Inc. Customer Support, they are documented fully in the Universal Products Utilities 3.2.0 User Guide.

To protect the database contents, operating system permissions on the database files themselves should be set so that only accounts with super-user or administrative privileges has access to them.

UEM provides its own command line utility, UEMLoad, to maintain the event definition and event handler databases. While the contents of these databases can be viewed using the Universal Spool List utility, it is recommended that all access be done using UEMLoad. The ability to remove event definition and event handler records is only provided with UEMLoad.

UEMLoad only can manage event definition and event handler databases that are local to the system on which it resides. To process a request, UEMLoad sends a request to the Universal Broker running on that system to start a demand-driven UEM Server. Next, UEMLoad sends the database request to the UEM Server, so that the UEM Server can validate the request and provide any required default values. The UEM Server then forwards the request to the Universal Broker, so that the changes can be applied to the appropriate database.

UEMLoad executes in the security context of the user account that started it. Since the Universal Broker applies changes to the event definition and event handler databases, any user with the authority to execute UEMLoad will, effectively, have access to a secure resource. It is therefore strongly recommended that the privileges on UEMLoad be set such that only those user accounts with super-user or administrative privileges be allowed to execute it.

Application support also is provided to further limit access to the event definition and event handler databases. A type of Universal Access Control List (UACL) is provided by UEM to grant or deny local user accounts the authority to access these databases.

To fully secure the event definition and event handler databases, a UACL entry can be defined to deny access to all user accounts. Then, additional entries can be defined to grant database access to those user accounts with the appropriate authority. Whenever UEMLoad is executed, the entries in the UACL will be checked. If a match cannot be found which indicates that the user account that started UEMLoad is allowed to access the database, the application will terminate with an error.

Section 2.7 Universal Access Control List provides a more thorough overview of the UACL feature. For information on the specific UACL used to control access to the event definition and event handler databases, see the DATABASE_MAINTENANCE_ACL configuration option in the Universal Event Monitor 3.2.0 Reference Guide.

The event spool records generated by a UEM Server only can be viewed with the Universal Spool List utility.

# 2.7  Universal Access Control List

This section provides a general overview of the Universal Access Control List (UACL) capabilities offered by UEM Server as an extra layer of security to some of the services it offers.

The UACL consists of one or more entries that provide a means by which certain application requests can be granted or denied to specified user accounts or originating systems. A UACL also can be used to govern authentication requirements for a specified local user account.

## 2.7.1  UACL Configuration

All UACL rules are defined in one file, `uacl.conf`, which is required in order for UEM Server to start. If no UACL rules are required for UEM, this file can be left empty.

The UACL file must reside in one of the platform-dependent directories specified in Section 2.2.5 Configuration File. If it is possible for configuration files to reside in multiple locations, the first `uacl.conf` file found is the one that is used.

The UACL file syntax is the same as all other Universal Products configuration files for UNIX and Windows (see Section 2.2.6 Configuration File Syntax for details). For specific keywords and values used for UEM Server UACL entries, see the Universal Event Monitor 3.2.0 Reference Guide.

| Windows |
|---------|

Although UACL entries in the `uacl.conf` file can be edited with any text editor (for example, Notepad), the Universal Configuration Manager application is the recommended way to set configuration options. The Universal Configuration Manager provides a graphical interface and context-sensitive help, and helps protect the integrity of the UACL entries (see Section 2.4 Universal Configuration Manager).

# 2.7.2  UACL Entries

UACL entries consist of a *type* and a *rule*.

- *type* identifies the secured Universal Products feature to which the UACL entry applies.
- *rule* specifies the attributes that determine when an ACL entry should be applied, and if applicable, whether access to the secured feature is allowed or denied.

The UACL entry types used by UEM are listed below. The keyword used to identify the UACL type in the UEM Server configuration is shown in parentheses.

Note:   There is no limit to the number of UACL entries that can be specified.

For detailed information on configuring UACL entries used by UEM, see Chapter 7 Universal Event Monitor Server.

### *Access ACL* (uem_access)

Used to grant or deny a UEM Manager access to a demand-driven UEM Server. Access is allowed or denied based on the following attributes:

- IP address or host name of the system on which the UEM Manager is executing
- ID of the user account with which the UEM Manager is executing
- ID of a user account that is defined to the local system on which the UEM Server is executing

**UNIX**

An Access ACL entry also can specify whether a local user account must be authenticated in order for the UEM Server to continue.

### *Event Handler ACL* (uem_handler)

Used to grant or deny a user account the authority to execute an event handler process.

**UNIX**

An event handler ACL entry also specifies whether or not the user account must be authenticated with a password before executing the event handler process.

### *Database Maintenance ACL* (uem_maintenance)

Used to grant or deny a user account the authority to access the event definition and event handler databases. The types of access permitted for a given user account may also be specified.

## 2.7.3  Controlling Access

The identity of the client application requesting access to a secure UEM feature is compared against the entries in a UACL to determine whether or not access to that feature is permitted. A client application's identity can be expressed in terms of the user account used to execute the application, or the IP address (obtained using TCP/IP programming interfaces) of the process's host system.

Some UACL rules, such as those that govern access to a demand-driven UEM Server, also provide the ability to secure the feature with a local user account. Specifying a local user account effectively restricts the security contexts in which a demand-driven UEM Server can be executed.

Table 2.2, below, describes the rules used for matching user IDs and host addresses during evaluation of a UACL.

| Attribute | Description |
|---|---|
| User Account | A user account can be used as an attribute in a UACL entry to control access based on a client or local user account. A client account is the ID of the user requesting access to the secure feature. A local account is the ID of the user in whose security context the feature will be executed. |
| | The following rules are used to determine a user account match: |
| | An asterisk ( **\*** ) matches 0 or more characters and a question mark ( **?** ) matches one character. For example, **AB\*M** matches **ABCDM** and **ABM**. **AB?M** matches **ABCM** but not **ABCDM**. |
| | To delimit a special character (for example:  **\***, **?** ) and cause it to be treated as a literal character during a comparison, prefix the character with a forward slash ( **/** ). For example: **A/\*B** matches **A\*B**.  **A//B** matches **A/B**. |
| | Adding the control codes **/c** or **/C** to the beginning of a user ID will cause a case-insensitive or case-sensitive comparison, respectively, to be done. By default, comparisons are case-sensitive. For example,  **/cABC** matches **abc**, **AbC**, and **aBc**. **/ca/Cbc** matches **Abc** and **abc**, but not **ABC**. |
| Host Name | A host name can be used as an attribute in a UACL entry to control access based on the IP address or alias of the system from which the request to access a secure feature was made. |
| | When using an IP address to identify a client, be sure to account for any Network Address Translation (NAT) being done that would change a client's actual IP address. In this situation, it would be necessary to use the translated IP address in the UACL rule. |
| | The following are acceptable host name values: |
| | A complete IP address in dotted decimal notation. For example, **10.20.30.40**. |
| | A prefix of the IP address in dotted decimal notation. For example, **10.20.30.** matches all IP addresses starting with **10.20.30.**. The last dot ( **.** ) is required. |
| | An IP network address and its corresponding subnet mask, specified in the format **netaddr/netmask**. For this comparison, a bitwise-logical AND of the remote IP address and the specified **netmask** is done. A match is made if the result is equal to **netaddr**. For example, **131.155.72.0/255.255.254.0** matches IP addresses in the range **131.155.72.0** through **131.155.73.255**. |
| | A Domain Name System (DNS) representation of the host's IP address. For example, **sysa.abc.com**. |
| | A DNS suffix that will be used to match a range of IP addresses. For example, **.abc.com** matches all host names ending with **abc.com**, such as **sysa.abc.com**. The first dot ( **.** ) is required. |
| | The literal **ALL**. This value matches all IP addresses. This string must be all UPPERCASE. |

Table 2.2  Rules Used During UACL Evaluation.

# 2.8  Message and Audit Facilities

UEM uses the same message facilities as all other Universal Products. For purposes of this discussion, messages include textual messages written to a console, file, or system log that records an application's actions and error conditions.

This section describes the message and audit facilities available in UEM.

## 2.8.1  Message Destination

The location to which messages are written is referred to as the message destination. For UEM components, the message destination depends on the host operating system. Valid destinations are described in the platform-specific sections below.

**z/OS**

The only UEM component available on z/OS is the UEM Manager, which runs as a batch job. All messages are written to the location referenced by the SYSOUT DD statement.

**UNIX**

The UEM Manager and UEMLoad write all messages to standard error.

The UEM Server sends its messages over a UDP connection to a local Universal Broker. The Universal Broker displays each message received in its log file, along with a date and time stamp and the UEM Server's process ID.

The Universal Broker's log file is named `unv.log`, and resides in `/var/opt/universal/log` by default.

**Windows**

The UEM Manager and UEMLoad write all messages to standard error.

The UEM Server sends its messages over a UDP connection to a local Universal Broker. All messages sent to and reported by the Universal Broker are written to the Windows Application Event Log.

## 2.8.2  Message Levels

Each message has a message-level attribute that indicates its severity. Possible severities, in ascending order of magnitude, are: Audit, Informational, Warning, and Error.

The MESSAGE_LEVEL configuration option is used to control which messages are reported by each UEM component.

Message levels have a defined relationship to each other. Messages with a severity at or above the specified message level are reported. Messages with a severity lower than the specified message level are suppressed.

Therefore:

- Message level of Audit causes all messages to be reported.
- Message level of Informational causes all Informational, Warning, and Error messages to be reported.
- Message level of Warning causes all Warning and Error messages to be reported.
- Message level or Error causes only Error messages to be reported.

Audit messages are those that document the configuration options used and resources allocated for the program's execution. This level provides complete description of the program execution for auditing and problem resolution.

Informational messages describe the actions being taken by a program. The messages help determine at what stage of processing a program may be. Informational messages also document data processing statistics.

Warning messages include those that document unexpected behavior that may or may not indicate a problem.

Error messages include those that document program errors. The message provides diagnostic data to help identify a problem's cause.

### Trace Message Level

An additional message level, Trace, is provided that allows detailed diagnostic information generated during a program's execution to be written to a text file. Because of the amount of information generated and the resulting impact on process performance, this message level should be used only following a recommendation from Stonebranch Inc. Customer Support. The information contained in a trace file may be the only means available to locate the cause of an application problem.

### Message Identifier

All messages are prefixed with a message identifier. The message ID format is `UNVnnnnl`, where `nnnn` is the message number and `l` is the message level. Possible values for `l` are (`A`)udit, (`I`)nformational, (`W`)arning, and (`E`)rror. All message ID's for UEM are documented in the Universal Products 3.2.0 Messages and Codes document.

# Chapter 3
# Universal Event Monitor Manager

## 3.1 Overview

The Universal Event Monitor Manager (`uem`) is provided for monitoring a single event. The parameters which define the system event and any actions that should be taken when the event satisfies certain conditions are specified via the UEM Manager's command options.

When a UEM Manager is started, it sends a request to the specified local or remote Universal Broker to start a Universal Event Monitor Server (`uemsrv`). The UEM Server that is started is a *demand-driven* Server, because it is executed *on demand* by a UEM Manager.

The UEM Manager sends the monitoring request to the UEM Server, which validates the provided parameters and supplies default values for any required parameters not specified.

After the request is validated and its receipt acknowledged by the UEM Server, the UEM Manager will wait (by default) for the UEM Server process to finish. This will occur when the following conditions are satisfied:

1. Event becomes inactive, which occurs either when:
   a. Required number of expected event occurrences are detected.
   b. Inactive date and time specified for the event elapses.
2. Any handler processes executed by the UEM Server have completed.

As noted in the conditions listed above, one or more handler processes can be executed by the UEM Server. The scenarios under which these processes are executed come from information provided by the UEM Manager when the actions to take for `trigger` or `reject` event occurrences are specified.

The actions to take when an event is set to an `expired` state also can be specified in the UEM Manager command statement.

Before it ends, the UEM Server may inform the UEM Manager of the result of the monitoring request. If it does, the UEM Manager will set its exit code based on this information. This is the default behavior. However, if an option was set in the UEM Manager instructing it to not wait on the UEM Server, the UEM Manager will end as soon as the UEM Server acknowledges its receipt of a valid monitoring request.

Detailed usage information, by operating system, for UEM Manager is included in the following chapters.

For detailed information on UEM configuration options, see the Universal Event Monitor 3.2.0 Reference Guide.

# Chapter 4
# Universal Event Monitor Manager for z/OS

## 4.1  Overview

This chapter provides information on Universal Event Monitor (UEM) Manager specific to the z/OS operating system.

UEM Manager monitors a single event on any computer running a UEM Server component. The event to monitor is described using an existing record in the event definition database or values supplied via command parameters. Likewise, parameters describing event handlers also can be supplied from an existing event handler record or from command line options.

A UEM Manager causes a demand-driven UEM Server component to be started on the target system. The UEM Server is classified as demand-driven because it is started upon demand by a UEM Manager.

A UEM Server is classified as event-driven when it is started automatically by the Universal Broker, without a request from a UEM Manager.

It is the demand-driven UEM Server that is responsible for monitoring the event and executing any processes on behalf of the specified event handlers. The UEM Manager may finish as soon as the UEM Server begins monitoring the event, or it may wait until the UEM Server completes, in which case the UEM Manager will receive status messages regarding monitoring activity.

# 4.2  Usage

UEM Manager for z/OS executes as a batch job.

This section describes the JCL, configuration and configuration options, and command line syntax of UEM Manager for z/OS.

Section 4.3 Examples of UEM Manager for z/OS provides examples demonstrating the flexibility of Universal Event Monitor.

## 4.2.1  JCL Procedure

Figure 4.1, below, illustrates the Universal Event Monitor for z/OS JCL procedure (**UEMPRC**, located in the **SUNVSAMP** library) that is provided to simplify the execution JCL and future maintenance.

```
//UEMPRC  PROC UPARM=,              -- UEM options
//             UEMPRE=#SHLQ.UNV
//*
//PS1      EXEC PGM=UEM,PARM='ENVAR(TZ=EST5EDT)/&UPARM'
//STEPLIB  DD  DISP=SHR,DSN=&UEMPRE..SUNVLOAD
//*
//UNVNLS   DD  DISP=SHR,DSN=&UEMPRE..SUNVNLS
//UNVTRACE DD  SYSOUT=*
//*
//SYSPRINT DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
```

Figure 4.1  UEM Manager for z/OS – JCL Procedure

The symbolic parameter **UPARM** is provided to allow EXEC PARM keyword values to be specified for the UEM program. The PARM values to the left of the slash ( **/** ) character are IBM Language Environment (LE) parameters. (See the Universal Products 3.2.0 Installation Guide for information regarding the customization of Language Environment parameters.)

The **CONFIG** symbolic parameter can be used to specify the name of the PDS member in which persistent configuration options for UEM Manager reside.

The **UEMPRE** symbolic parameter specifies the data set name prefix of Universal Products installation data sets.

## 4.2.2  DD Statements used in JCL Procedure

Table 4.1, below, describes the DD statements used in the Universal Event Manager for z/OS JCL procedure illustrated in Figure 4.1.

| ddname | DCB Attributes * | Mode | Description |
|--------|------------------|------|-------------|
| STEPLIB | DSORG=PO, RECFM=U | input | Universal Products load library which contains the program to execute. |
| UNVNLS | DSORG=PO, RECFM=(F, FB, V, VB) | input | Universal Products national language support library. Contains message catalogs and code page translation tables. |
| UNVTRACE | DSORG=PS, RECFM=(F, FB, V, VB) | output | UEM trace output. |
| SYSPRINT | DSORG=PS, RECFM=(F, FB, V, VB) | output | Standard output file for the UEM program. UEM does not write any messages to SYSPRINT. |
| SYSOUT | DSORG=PS, RECFM=(F, FB, V, VB) | output | Standard error file for the UEM program. UEM writes its messages to SYSOUT. |
| * The C runtime library determines the default DCB attributes.  Refer to the IBM manual *OS/390 C/C++ Programming Guide* for details on default DCB attributes for stream I/O | | | |

Table 4.1  UEM Manager for z/OS –  DD Statements in JCL Procedure

## DD Statement Categories

UEM Manager for z/OS DD statements are organized into the following categories:

- Runtime specifications
  STEPLIB, UNVCONF, and UNVNLS are in this category.
- UEM message and command files
  SYSPRINT, SYSOUT, and UNVTRACE are in this category.
- Command files
  User-defined DD statements to which Universal Event Monitor commands refer.

# 4.2.3  JCL

Figure 4.2, below, illustrates the Universal Event Monitor for z/OS JCL using the **UEMPRC** procedure illustrated in Figure 4.1.

```
//jobname JOB CLASS=A,MSGCLASS=X
//STEP1 EXEC UEMPRC
//ENCFILE DD DISP=SHR,DSN=UEM.UENCRYPT.USERINFO(MYHOST)
//SYSIN   DD *
 -event_type file -filespec "myfile*.dat" -host myhost
 -encryptedfile ENCFILE
/*
```

Figure 4.2  UEM Manager for z/OS – JCL

Job step STEP1 executes the procedure **UEMPRC**.

The command options are specified on the SYSIN DD.

In this example, a UEM Server is executed on the remote host **myhost** in order to detect the occurrence of a file that matches the file specification of **myfile\*.dat**. The contents of **ENCFILE** include an encrypted user ID and password that are validated on the remote host.

## 4.2.4  Configuration

Configuration consists of:

- Setting default options and preferences for all executions of UEM Manager.
- Setting options and preferences for a single execution of UEM Manager.

UEM for z/OS receives its configuration options from the following sources:

1. PARM keyword
2. SYSIN DD statement
3. Command file
4. Configuration file

The order of precedence is the same as the list above; PARM keyword options being the highest and configuration file being the lowest. That is, options specified via a PARM keyword override options specified via a SYSIN ddname, and so on.

The configuration file provides the simplest method of specifying configuration options whose values will not change with each command invocation. These default values are used if the options are not read from one or more other sources.

Some options only can be specified in the configuration file; they have no corresponding command line equivalent. Other options cannot be specified in the configuration file; they must be specified via one or more other sources for a single execution of UEM Manager.

For UEM Manager for z/OS, configuration options are placed in the configuration file that is referenced in the `UEMPRC` JCL procedure by the `UNVCONF` DD statement, member `UEMCFG00`.

The short and long forms of configuration options are used when an option is specified in the SYSIN DD statement.

- Long form consists of two or more case-insensitive characters; it is available for all command options.
- Short form consists of a single case-sensitive character; it is available for some command options.

See Section 2.2.1 Configuration Methods for complete details on configuration methods and command input for Universal Products.

# 4.2.5  Configuration Options

This section describes the configuration options used to execute UEM Manager for z/OS.

## Configuration Options Categories

Table 4.2, below, categorizes the configuration options according to function.

| Category | Description |
|----------|-------------|
| Event Definition | System event to monitor. |
| Event Handler | Actions that should be taken when an event occurrence is triggered or rejected, or when an event expires. |
| Local | Options required for local broker registration. |
| Message | Universal Event Monitor message options. |
| Miscellaneous | Display command help and program versions. |
| Monitoring | Options that control how an event is monitored by the remote UEM Server. |
| Network | Transferring data between the local and remote systems. |
| Options | Alternative methods available for specifying command options. |
| Remote | Network address of the remote system and connection options. |
| User | Identify the user account with which monitoring activity is conducted on the remote system. |

Table 4.2  UEM Manager for z/OS - Configuration Options Categories

The UEM Manager configuration options for each category are summarized in the following tables. Each **Option Name** is a link to detailed information about that option in the Universal Event Monitor 3.2.0 Reference Guide.

## Event Definition Category Options

| Option Name | Description |
|-------------|-------------|
| EVENT_ID | ID of a stored event definition record. |
| EVENT_TYPE | Type of event to monitor. |
| HANDLER_OPTIONS | Options that are passed as command line arguments to all processes executed on behalf of an event handler. See OPTIONS for handler-specific command line options. |
| INACTIVE_DATE_TIME | Date and time at which the state of the monitored event should be made "inactive." |
| TRACKING_INTERVAL | Frequency, in seconds, with which a tracked event occurrence is tested for completeness. |

**Event Definition Category Options - Type-Specific**

These options are specific to event definitions with an EVENT_TYPE of `FILE`.

| Option Name | Description |
|---|---|
| FILE_SPECIFICATION | Name or pattern of the file whose creation should be detected and tracked for completion. |
| MINIMUM_FILE_SIZE | Smallest size a file may be in order for it to be considered complete. |
| RENAME_FILE | Flag that indicates whether or not a completed file should be renamed. |
| RENAME_FILE_SPECIFICATION | Name or pattern to use when a file is renamed. |

## Event Handler Category Options

| Option Name | Description |
|---|---|
| EVENT_STATE | Event state that, when encountered, will result in the execution of the associated event handler. |
| HANDLER_ID | ID of a stored event handler record. |
| HANDLER_TYPE | Type of process to execute. Reserved for future integration with other Universal Product applications. |
| MAXIMUM_RETURN_CODE | Highest return code that an event handler can exit with to be considered as having executed successfully. |
| OPTIONS | Values that are passed as command line arguments to a particular handler specified for a given EVENT_STATE. |
| USER_COMMAND | Complete path to an application file or remote script that should be executed on behalf of the event handler. |
| USER_SCRIPT | USER_SCRIPT DD statement that contains one or more system commands that should be executed on behalf of the event handler |
| USER_SCRIPT_TYPE | Type of script interpreter used to evaluate and execute the commands contained in USER_SCRIPT. |

## Local Category Options

| Option Name | Description |
|---|---|
| SYSTEM_ID | Local Universal Broker with which the UEM Manager must register. |

## Message Category Options

| Option Name | Description |
|---|---|
| MESSAGE_LANGUAGE | Language of messages written. |
| MESSAGE_LEVEL | Level of messages written. |
| TRACE_FILE_LINES | Maximum number of lines written to a trace file before it wraps around. |
| TRACE_TABLE | Memory trace table specification. |

## Miscellaneous Category Options

| Option Name | Description |
|---|---|
| HELP | Write command option help. |
| VERSION | Write program version. |

## Monitoring Category Options

| Option Name | Description |
|---|---|
| MAX_OCCURRENCE_COUNT | Maximum number of event occurrences to monitor. |
| POLLING_INTERVAL | Frequency with which the UEM Server will detect new occurrences of the system event. The UEM Server will also check at this time to see if the monitored event should be made inactive. |
| WAIT | Forces the UEM Manager to wait for the completion of the UEM Server. |

## Network Category Options

| Option Name | Description |
|---|---|
| CODE_PAGE | Code page used for text translation. |
| CTL_SSL_CIPHER_LIST | SSL cipher list for the control session established between the UEM Manager and Server. |
| NETWORK_DELAY | Maximum number of seconds to wait for data communications. |

## Options Category Options

| Option Name | Description |
|---|---|
| COMMAND_FILE_ENCRYPTED | Encrypted file that contains some command options. |
| COMMAND_FILE_PLAIN | Plain text file that contains some command options. |
| COMMAND_ID | ID that identifies unit of work represented by the UEM Manager and its associated UEM Server |
| ENCRYPTION_KEY | Optional encryption key used to decrypt the encrypted command file specified by the COMMAND_FILE_ENCRYPTED option. |

## Remote Category Options

| Option Name | Description |
|---|---|
| CONNECT_TIMEOUT | Amount of time that a UEM Manager will wait for a connection to a remote Universal Broker to complete. |
| DNS_EXPAND | Number of IP addresses returned to UEM Manager following a DNS query issued to resolve a host name. |
| HOST_SELECTION | Host in the REMOTE_HOST list that the UEM Manager will choose to begin its attempts to connect to a remote Universal Broker. |
| HOSTNAME_RETRY_COUNT | Maximum number of attempts that will be made to establish a connection with the remote host. |
| OUTBOUND_IP | Host or IP address to use for all outgoing IP connections. |
| REMOTE_HOST | List of one or more hosts upon which a command may run. |
| REMOTE_PORT | TCP/IP port number on which the Universal Broker is accepting connections. |

## User Category Options

| Option Name | Description |
|---|---|
| LOGIN | Instruction for the UEM Server to establish an execution environment for a user account. |
| USER_ID | ID of a remote user account that the UEM Server uses to establish the security context in which event monitoring is performed. |
| USER_PASSWORD | Password associated with USER_ID. |

# 4.2.6  Command Line Syntax

Figure 4.3, below, illustrates the command line syntax — using the command line, long form of the configuration options — of UEM Manager for z/OS.

```
uem
-host hostlist
[-connect_timeout seconds]
[-dns_expand {yes|no}]
[-host_selection {sequential|random}]
[-port port]
[-system_id ID]
[-userid user [-pwd password] ]
[-login {yes|no}]
[-codepage codepage]
[-file dataset | -encryptedfile ddname [-key key] ]   *
[-cmdid id]
[-ctl_ssl_cipher_list cipherlist]
{-event_id id | -event_type type -filespec filespecification
    [-min_file_size size[{b|k|m|g}] ] [-rename_file {yes|no}]
    [-rename_filespec renamespecification] }
[ {-triggered | -rejected | -expired
    {-handler_id id | -cmd command | -script ddname [-script_type type] }
    [-options options] [-handler_type {cmd|script}] [-maxrc returncode] } ]
[-handler_opts options]
[-hostname_retry_count count]
[-inact_date_time time]
[-max_count count]
[-lang language]
[-level {trace|audit|info|warn|error}[,{time|notime}] ]
[-delay seconds]
[-outboundip host]
[-polling_int seconds]
[-tracefilelines lines]
[-trace_table size[{b|k|m|g}] [,{error|always|never}] ]
[-tracking_int seconds]
[-wait {yes|no}]

uem
{-help | -version}
```

\* The command file (-file or -encryptedfile) can contain some or all required and/or optional configuration options, including -host. If a command file is specified on the command line, and it contains the required -host option, that option does not have to be specified additionally on the command line.

Figure 4.3  Universal Event Monitor Manager for z/OS - Command Line Syntax)

# 4.3  Examples of UEM Manager for z/OS

Appendix A Examples provides operating system-specific examples that demonstrate the use of UEM.

Included in this appendix are the following examples that demonstrate the use of UEM Manager for z/OS:

- Starting a UEM (Event-Driven) Server
- Refreshing a UEM Server (Event-Driven)
- Using a Stored Event Handler Record
- Handling an Event With a Script
- Handling an Expired Event
- Continuation Character "-" in z/OS Handler Script
- Continuation Character "+" in z/OS Handler Script
- Continuation Characters "-" and "+" in z/OS Handler Script

# 4.4  Security

The UEM Manager is designed to be a secure system.

As the level of security rises, so does the administrative complexity of the system. The UEM Manager has balanced the two to avoid the administrative complexity with a minimum sacrifice to security.

Security issues addressed by UEM Manager include:

1. Access to Universal Event Monitor's files
2. Privacy and integrity of transmitted network data
3. RACF access to the remote system with a specific user identity

## 4.4.1  Data Set Permissions

Only trusted user accounts should have write access to the UEM Manager installation files. Eligible users of UEM require read access to the national language support library **SUNVNLS**, the configuration file **UNVCONF**, and the load library **SUNVLOAD**.

## 4.4.2  Data Privacy

Data transmitted from a UEM Manager across a network connection to the Universal Broker and demand-driven UEM Server is protected using features present in all Stonebranch Inc. Universal Products.

For more information on the steps taken to protect transferred data, see Section 2.5.1 Security.

## 4.4.3  RACF Protection

The UEM Manager verifies a user's access to a RACF general resource profile. The resource profile controls a user's ability to monitor an event on a remote host with a specific remote user identity.

See the Universal Products 3.2.0 Installation Guide for complete details on installing and administering UEM Manager RACF profiles.

# Chapter 5
# Universal Event Monitor Manager
# for Windows

## 5.1 Overview

This chapter provides information on Universal Event Monitor (UEM) Manager specific to the Windows operating system.

A UEM Manager monitors a single event on any computer running a UEM Server component. The event to monitor is described using an existing record in the event definition database or values supplied via command line parameters. Likewise, parameters describing event handlers also can be supplied from an existing event handler record or from command line options.

A UEM Manager causes a demand-driven UEM Server component to be started on the target system. The UEM Server is classified as demand-driven because it is started upon demand by a UEM Manager. A UEM Server is classified as event-driven when it is started automatically by the Universal Broker, without a request from a UEM Manager.

It is the demand-driven UEM Server that is responsible for monitoring the event and executing any processes on behalf of the event handlers. The UEM Manager can finish as soon as the UEM Server begins monitoring the event, or it can wait until the UEM Server completes, in which case the UEM Manager will receive status messages regarding monitoring activity.

# 5.2  Usage

UEM Manager for Windows executes as a command line application.

This section describes the configuration, configuration options, and command line syntax of UEM Manager for Windows.

Section 5.3 Examples of UEM Manager for Windows provides examples demonstrating the flexibility of Universal Event Monitor.

## 5.2.1  Configuration

Configuration consists of:

- Setting default options and preferences for all executions of UEM Manager.
- Setting options and preferences for a single execution of UEM Manager.

UEM for Windows receives its configuration options from the following sources:

1. Command line
2. Command file
3. Environment variables
4. Configuration file

The order of precedence is the same as the list above; command line options being the highest and configuration file being the lowest. That is, options specified via a command line override options specified a via command file, and so on.

The configuration file provides the simplest method of specifying configuration options whose values will not change with each command invocation. These default values are used if the options are not read from one or more other sources.

Some options only can be specified in the configuration file; they have no corresponding command line equivalent. Other options cannot be specified in the configuration file; they must be specified via one or more other sources for a single execution of UEM Manager.

See Section 2.2.1 Configuration Methods for complete details on configuration methods and command input for Universal Products.

## 5.2.2  Configuration Options

This section describes the configuration options used to execute UEM Manager for Windows.

## Configuration Options Categories

Table 5.1, below, categorizes the configuration options according to function.

| Category | Description |
|---|---|
| Event Definition | Describe the system event to monitor. |
| Event Handler | Describe the actions that should be taken when an event occurrence is triggered or rejected, or when an event is set to an expired state. |
| Installation | Options that specify installation requirements, such as directory locations |
| Local | Options required for local broker registration. |
| Message | UEM message options. |
| Miscellaneous | Display command help and program versions. |
| Monitoring | Control how an event is monitored by the remote UEM Server. |
| Network | Transferring data between the remote and local systems. |
| Options | Alternative methods available for specifying command options. |
| Remote | Network address of the remote system and connection options. |
| User | User account with which monitoring activity is conducted on the remote system. |

Table 5.1  UEM Manager for Windows - Configuration Options Categories

The UEM Manager configuration options for each category are summarized in the following tables. Each **Option Name** is a link to detailed information about that option in the Universal Event Monitor 3.2.0 Reference Guide.

## Event Definition Category Options

| Option Name | Description |
|---|---|
| EVENT_ID | ID of a stored event definition record. |
| EVENT_TYPE | Type of event to monitor. |
| HANDLER_OPTIONS | Options that are passed as command line arguments to all processes executed on behalf of an event handler. See OPTIONS for handler-specific command line options. |
| HANDLER_OPTIONS | Date and time at which the state of the monitored event should be made "inactive". |
| TRACKING_INTERVAL | Frequency, in seconds, with which a tracked event occurrence is tested for completeness. |

### Event Definition Category Options - Type-Specific

These options are specific to event definitions with an EVENT_TYPE of `FILE`.

| Option Name | Description |
| --- | --- |
| FILE_SPECIFICATION | Name or pattern of the file whose creation should be detected and tracked for completion. |
| MINIMUM_FILE_SIZE | Smallest size a file can be in order for it to be considered complete. |
| RENAME_FILE | Flag that indicates whether or not a completed file should be renamed. |
| RENAME_FILE_SPECIFICATION | Name or pattern to use when a file is renamed. |

## Event Handler Category Options

| Option Name | Description |
| --- | --- |
| EVENT_STATE | Describes the event state that, when encountered, will result in the execution of the associated event handler. |
| HANDLER_ID | ID of a stored event handler record. |
| HANDLER_TYPE | Type of process to execute. Reserved for future integration with other Universal Product applications. |
| MAXIMUM_RETURN_CODE | Highest return code that an event handler can exit with to be considered as having executed successfully. |
| OPTIONS | Values that are passed as command line arguments to a particular handler specified for a given EVENT_STATE. |
| USER_COMMAND | Complete path to an application file or remote script that should be executed on behalf of the event handler. |
| USER_SCRIPT | Complete path to a local script file that contains one or more system commands that should be executed on behalf of the event handler. |
| USER_SCRIPT_TYPE | Describes the type of script interpreter used to evaluate and execute the commands contained in USER_SCRIPT. |

## Installation Category Options

| Option Name | Description |
| --- | --- |
| INSTALLATION_DIRECTORY | Directory in which UEM Manager is installed. |

## Message Category Options Summary

| Option Name | Description |
| --- | --- |
| MESSAGE_LANGUAGE | Language of messages printed. |
| MESSAGE_LEVEL | Level of messages printed. |
| NLS_ DIRECTORY | Directory location of message catalog and code page tables |
| TRACE_FILE_LINES | Maximum number of lines written to a trace file before it wraps around. |
| TRACE_TABLE | Memory trace table specification. |

## Miscellaneous Category Options Summary

| Option Name | Description |
| --- | --- |
| HELP | Write command option help. |
| VERSION | Write program version. |

## Monitoring Category Options Summary

| Option Name | Description |
| --- | --- |
| MAX_OCCURRENCE_COUNT | Maximum number of event occurrences to monitor. |
| POLLING_INTERVAL | Frequency with which the UEM Server will detect new occurrences of the system event. The UEM Server will also check at this time to see if the monitored event should be made inactive. |
| WAIT | Forces the UEM Manager to wait for the completion of the UEM Server. |

## Network Category Options Summary

| Option Name | Description |
| --- | --- |
| CODE_PAGE | Code page used for text translation. |
| CTL_SSL_CIPHER_LIST | SSL cipher list for the control session. |
| NETWORK_DELAY | Maximum number of seconds considered acceptable to wait for data communications. |

## Options Category Options Summary

| Option Name | Description |
| --- | --- |
| COMMAND_FILE_ENCRYPTED | Encrypted file that contains command options. |
| COMMAND_FILE_PLAIN | Plain text file that contains command options. |
| COMMAND_ID | ID that identifies unit of work represented by the UEM Manager and its associated UEM Server |
| ENCRYPTION_KEY | Optional encryption key used to decrypt the encrypted user file specified by the COMMAND_FILE_ENCRYPTED option. |

## Remote Category Options Summary

| Option Name | Description |
| --- | --- |
| CONNECT_TIMEOUT | Amount of time that a UEM Manager will wait for a connection to a remote Universal Broker to complete. |
| DNS_EXPAND | Number of IP addresses returned to UEM Manager following a DNS query issued to resolve a host name. |
| HOST_SELECTION | Host in the REMOTE_HOST list that the UEM Manager will choose to begin its attempts to connect to a remote Universal Broker. |
| HOSTNAME_RETRY_COUNT | Maximum number of attempts that will be made to establish a connection with the remote host. |
| OUTBOUND_IP | Host or IP address to use for all outgoing IP connections. |
| REMOTE_HOST | List of one or more hosts upon which a command may run. |
| REMOTE_PORT | TCP/IP port number on which the Universal Broker is accepting connections. |

## User Category Options Summary

| Option Name | Description |
| --- | --- |
| LOGIN | Instruction for the UEM Server to establish an execution environment for a user account. |
| USER_ID | ID of a remote user account that the UEM Server uses to establish the security context in which event monitoring is performed. |
| USER_PASSWORD | Password associated with USER_ID. |

# 5.2.3  Command Line Syntax

Figure 5.1, below, illustrates the command line syntax — using the command line, long form of the configuration options — of UEM Manager for Windows.

```
uem
-host hostlist
[-connect_timeout seconds]
[-dns_expand {yes|no}]
[-host_selection {sequential|random}]
[-port port]
[-userid user [-pwd password] ]
[-login {yes|no}]
[-codepage codepage]
[-file filename | -encryptedfile filename [-key key] ]   *
[-cmdid id]
[-ctl_ssl_cipher_list cipherlist]
{-event_id id | -event_type type -filespec filespecification
   [-min_file_size size[{b|k|m|g}] ] [-rename_file {yes|no}]
   [-rename_filespec renamespecification] }
[ {-triggered | -rejected | -expired
   {-handler_id id | -cmd command | -script filename [-script_type type] }
   [-options options] [-handler_type {cmd|script}] [-maxrc returncode] } ]
[-handler_opts options]
[-hostname_retry_count count]
[-inact_date_time time]
[-max_count count]
[-lang language]
[-level {trace|audit|info|warn|error}[,{time|notime}] ]
[-delay seconds]
[-outboundip host]
[-polling_int seconds]
[-tracefilelines lines]
[-trace_table size[{b|k|m|g}] [,{error|always|never}] ]
[-tracking_int seconds]
[-wait {yes|no}]


uem
{-help | -version}
```

> **\***  The command file (`-file` or `-encryptedfile`) can contain some or all required and/or optional configuration options, including `-host`. If a command file is specified on the command line, and it contains the required `-host` option, that option does not have to be specified additionally on the command line.

Figure 5.1  Universal Event Monitor Manager for Windows - Command Line Syntax

# 5.3  Examples of UEM Manager for Windows

Appendix A Examples provides operating system-specific examples that demonstrate the use of UEM.

Included in this appendix are the following examples that demonstrate the use of UEM Manager for Windows:

- Using a Stored Event Handler Record
- Executing a Script for a Triggered Event Occurrence
- Handling an Expired Event

# 5.4  Security

UEM Manager for Windows is designed to be a secure system. As the level of security rises, so does the administrative complexity of the system. UEM Manager has balanced the two to avoid the administrative complexity with a minimum sacrifice to security.

Security issues addressed by UEM Manager include:

- Access to UEM's files.
- Access to UEM configuration options.
- Privacy and integrity of transmitted network data.

## 5.4.1  File Permissions

Only trusted user accounts, which are most likely those that are members of the Administrators group, should be granted write access to the UEM Manager installation directory and subdirectories, and the files within them.

Authorized users of UEM require read access to the message catalogs (`*.umc` files), which reside in the `.\Universal\nls` directory. If UEM Manager is installed on an NTFS partition, these file permissions are set automatically during the installation.

## 5.4.2  Universal Configuration Manager

To protect access to UEM Manager configuration settings, the Universal Configuration Manager can be executed only by accounts in the Administrator group.

For more information on the Universal Configuration Manager, see Section 2.4 Universal Configuration Manager.

## 5.4.3  Data Privacy

Data transmitted from a UEM Manager across a network connection to the Universal Broker and demand-driven UEM Server is protected using features present in all Stonebranch Inc. Universal Products.

For more information on the steps taken to protect transferred data, see Section 2.5.1 Security.

# Chapter 6
# Universal Event Monitor Manager for UNIX

## 6.1 Overview

This chapter provides information on Universal Event Monitor (UEM) Manager specific to the UNIX operating system.

A UEM Manager monitors a single event on any computer running a UEM Server component. The event to monitor is described using an existing record in the event definition database or values supplied via command parameters. Likewise, parameters describing event handlers also can be supplied from an existing event handler record or from command line options.

A UEM Manager causes a demand-driven UEM Server component to be started on the target system. The UEM Server is classified as demand-driven because it is started upon demand by a UEM Manager. A UEM Server is classified as event-driven when it is started automatically by the Universal Broker, without a request from a UEM Manager.

It is the demand-driven UEM Server that is responsible for monitoring the event and executing any processes on behalf of the event handlers. The UEM Manager may finish as soon as the Server begins monitoring the event, or it may wait until the UEM Server completes, in which case the Manager will receive status messages regarding monitoring activity.

# 6.2  Usage

UEM Manager for UNIX executes as a command line application.

This section describes the configuration, configuration options, and command line syntax of UEM Manager for UNIX.

Section 6.3 Examples of UEM Manager for UNIX provides examples demonstrating the flexibility of Universal Event Monitor.

## 6.2.1  Configuration

Configuration consists of:

- Setting default options and preferences for all executions of UEM Manager.
- Setting options and preferences for a single execution of UEM Manager.

UEM Manager for UNIX receives its configuration options from the following sources:

1. Command line
2. Command file
3. Environment variables
4. Configuration file

The order of precedence is the same as the list above; command line options being the highest and configuration file being the lowest. That is, options specified via a command line override options specified via a command file, and so on.

The configuration file, `uem.conf`, provides the simplest method of specifying configuration options whose values will not change with each command invocation. These default values are used if the options are not read from one or more other sources.

Some options only can be specified in the configuration file; they have no corresponding command line equivalent. Other options cannot be specified in the configuration file; they must be specified via one or more other sources for a single execution of UEM Manager.

See Section2.2.1 Configuration Methods for complete details on configuration methods and command input for Universal Products.

# 6.2.2  Configuration Options

This section describes the configuration options used to execute UEM Manager for UNIX.

## Configuration Options Categories

Table 6.1, below, categories the configuration options according to function.

| Category | Description |
|---|---|
| Event Definition | Describe the system event to monitor. |
| Event Handler | Describe the actions that should be taken when an event occurrence is triggered or rejected, or when an event is set to an expired state. |
| Installation | Options that specify installation requirements, such as directory locations |
| Local | Options required for local broker registration. |
| Message | UEM message options. |
| Miscellaneous | Display command help and program versions. |
| Monitoring | Control how an event is monitored by the remote UEM Server. |
| Network | Transferring data between the local and remote systems. |
| Options | Alternative methods available for specifying command options. |
| Remote | Network address of the remote system and connection options. |
| User | Identify the user account with which monitoring activity is conducted on the remote system. |

Table 6.1  UEM Manager for UNIX - Configuration Options Categories

The UEM Manager configuration options for each category are summarized in the following tables. Each **Option Name** is a link to detailed information about that option in the Universal Event Monitor 3.2.0 Reference Guide.

## Event Definition Category Options

| Option Name | Description |
|---|---|
| EVENT_ID | ID of a stored event definition record. |
| EVENT_TYPE | Type of event to monitor. |
| HANDLER_OPTIONS | Options that are passed as command line arguments to all processes executed on behalf of an event handler. See OPTIONS for handler-specific command line options. |
| INACTIVE_DATE_TIME | Date and time at which the state of the monitored event should be made "inactive". |
| TRACKING_INTERVAL | Frequency, in seconds, with which a tracked event occurrence is tested for completeness. |

### Event Definition Category Options - Type-Specific

These options are specific to event definitions with an EVENT_TYPE of **FILE**.

| Option Name | Description |
| --- | --- |
| FILE_SPECIFICATION | Name or pattern of the file whose creation should be detected and tracked for completion. |
| MINIMUM_FILE_SIZE | Smallest size a file may be in order for it to be considered complete. |
| RENAME_FILE | Flag that indicates whether or not a completed file should be renamed. |
| RENAME_FILE_SPECIFICATION | Name or pattern to use when a file is renamed. |

## Event Handler Category Options

| Option Name | Description |
| --- | --- |
| EVENT_STATE | Describes the event state that, when encountered, will result in the execution of the associated event handler. |
| HANDLER_ID | ID of a stored event handler record. |
| HANDLER_TYPE | Type of process to execute. Reserved for future integration with other Universal Product applications. |
| MAXIMUM_RETURN_CODE | Highest return code that an event handler may exit with to be considered as having executed successfully. |
| OPTIONS | Values that are passed as command line arguments to a particular handler specified for a given EVENT_STATE. |
| USER_COMMAND | Complete path to an application file or script that should be executed on behalf of the event handler. |
| USER_SCRIPT | Complete path to a local script file that contains one or more system commands that should be executed on behalf of the event handler. |
| USER_SCRIPT_TYPE | Describes the type of script interpreter used to evaluate and execute the commands contained in USER_SCRIPT_TYPE. |

## Installation Category Options

| Option Name | Description |
| --- | --- |
| INSTALLATION_DIRECTORY | Directory in which UEM Manager is installed. |

## Local Category Options

| Option Name | Description |
| --- | --- |
| BIF_DIRECTORY | Broker Interface File (BIF) directory where the Universal Broker interface file is located. |
| PLF_DIRECTORY | Program Lock File (PLF) directory where the program lock files are located. |

## Message Category Options

| Option Name | Description |
| --- | --- |
| MESSAGE_LANGUAGE | Language of messages printed. |
| MESSAGE_LEVEL | Level of messages printed. |
| NLS_ DIRECTORY | Directory location of message catalog and code page tables |
| TRACE_FILE_LINES | Maximum number of lines written to a trace file before it wraps around. |
| TRACE_TABLE | Memory trace table specification. |

## Miscellaneous Category Options

| Option Name | Description |
| --- | --- |
| HELP | Write command option help. |
| VERSION | Write program version. |

## Monitoring Category Options

| Option Name | Description |
| --- | --- |
| MAX_OCCURRENCE_COUNT | Maximum number of event occurrences to monitor. |
| POLLING_INTERVAL | Frequency with which the UEM Server will detect new occurrences of the system event. The UEM Server will also check at this time to see if the monitored event should be made inactive. |
| WAIT | Instructs the UEM Manager to wait for the completion of the UEM Server. |

## Network Category Options

| Option Name | Description |
| --- | --- |
| CODE_PAGE | Code page used for text translation. |
| CTL_SSL_CIPHER_LIST | SSL cipher list for the control session. |
| NETWORK_DELAY | Maximum number of seconds considered acceptable to wait for data communications. |

## Options Category Options

| Option Name | Description |
| --- | --- |
| COMMAND_FILE_ENCRYPTED | Encrypted file that contains command options. |
| COMMAND_FILE_PLAIN | Plain text file that contains command options. |
| COMMAND_ID | ID that identifies unit of work represented by the UEM Manager and its associated UEM Server |
| ENCRYPTION_KEY | Optional encryption key used to decrypt the encrypted user file specified by the COMMAND_FILE_ENCRYPTED option. |

## Remote Category Options

| Option Name | Description |
| --- | --- |
| CONNECT_TIMEOUT | Amount of time that a UEM Manager will wait for a connection to a remote Universal Broker to complete. |
| DNS_EXPAND | Number of IP addresses returned to UEM Manager following a DNS query issued to resolve a host name. |
| HOST_SELECTION | Host in the REMOTE_HOST list that the UEM Manager will choose to begin its attempts to connect to a remote Universal Broker. |
| HOSTNAME_RETRY_COUNT | Maximum number of attempts that will be made to establish a connection with the remote host. |
| OUTBOUND_IP | Host or IP address to use for all outgoing IP connections. |
| REMOTE_HOST | List of one or more hosts upon which a command may run. |
| REMOTE_PORT | TCP/IP port number on which the Universal Broker is accepting connections. |

## User Category Options

| Option Name | Description |
| --- | --- |
| LOGIN | Instruction for the UEM Server to establish an execution environment for a user account. |
| USER_ID | ID of a remote user account that the UEM Server uses to establish the security context in which event monitoring is performed. |
| USER_PASSWORD | Password associated with USER_ID. |

## 6.2.3  Command Line Syntax

Figure 6.1, below, illustrates the command line syntax — using the command line, long form of the configuration options — of UEM Manager for UNIX.

```
uem
-host hostlist
[-connect_timeout seconds]
[-dns_expand {yes|no}]
[-host_selection {sequential|random}]
[-port port]
[-userid user [-pwd password] ]
[-login {yes|no}]
[-bif_directory directory]
[-plf_directory directory]
[-codepage codepage]
[-file filename | -encryptedfile filename [-key key] ]  *
[-cmdid id]
[-ctl_ssl_cipher_list cipherlist]
{-event_id id | -event_type type -filespec filespecification
    [-min_file_size size[{b|k|m|g}] ] [-rename_file {yes|no}]
    [-rename_filespec renamespecification] }
[ {-triggered | -rejected | -expired
    {-handler_id id | -cmd command | -script filename [-script_type type] }
    [-options options] [-handler_type {cmd|script}] [-maxrc returncode] } ]
[-handler_opts options]
[-hostname_retry_count count]
[-inact_date_time time]
[-max_count count]
[-lang language]
[-level {trace|audit|info|warn|error}[,{time|notime}] ]
[-delay seconds]
[-outboundip host]
[-polling_int seconds]
[-tracefilelines lines]
[-trace_table size[{b|k|m|g}] [,{error|always|never}] ]
[-tracking_int seconds]
[-wait {yes|no}]

uem
{-help | -version}
```

Figure 6.1  Universal Event Monitor Manager for UNIX - Command Line Syntax

# 6.3  Examples of UEM Manager for UNIX

Appendix A Examples provides operating system-specific examples that demonstrate the use of UEM.

Included in this appendix are the following examples that demonstrate the use of UEM Manager for UNIX:

- Using a Stored Event Handler Record
- Executing a Script for a Triggered Event Occurrence
- Handling an Expired Event

# 6.4  Security

UEM Manager for UNUX is designed to be a secure system. As the level of security rises, so does the administrative complexity of the system. UEM Manager has balanced the two to avoid the administrative complexity with a minimum sacrifice to security.

Security issues addressed by UEM Manager include:

1. Access to UEM's files
2. Access to UEM configuration options
3. Privacy and integrity of transmitted network data

## 6.4.1  File Permissions

Only trusted user accounts should be given write access to the UEM Manager installation directory and subdirectories, and the files within them. Authorized users of UEM require read access to the message catalogs (`*.umc` files) in the `nls` subdirectory of the primary Universal Products installation directory.

## 6.4.2  Configuration Files

Only trusted user accounts should have write access to the UEM Manager's configuration file.

For more information on storing configuration options, see Section 2.2 Configuration.

## 6.4.3  Data Privacy

Data transmitted from a UEM Manager across a network connection to the Universal Broker and demand-driven UEM Server is protected using features present in all Stonebranch Inc. Universal Products.

For more information on the steps taken to protect transferred data, see Section 2.5.1 Security.

# Chapter 7
# Universal Event Monitor
# Server

## 7.1 Overview

Universal Event Monitor (UEM) Server is a Universal Broker-managed component whose primary functions include:

- Monitoring of system events described by event definitions
- Responding to different event outcomes
- Executing processes on behalf of event handlers
- Validating database maintenance requests received from the UEMLoad utility

An event definition describes a system event whose occurrence should be detected and monitored by a UEM Server. Event definition parameters are supplied to UEM Server by way of command options used to execute a UEM Manager, or from records stored in the event definition database.

UEM Server handles (that is, responds to) an event based on the result of monitoring system activity. For each defined system event monitored by UEM Server, zero to many occurrences of that event actually can be detected and tracked by UEM.

- If a UEM Server detects an occurrence of an event and later determines that the event occurrence has completed, UEM will set that occurrence to a `triggered` state.
- If a UEM Server detects an occurrence of an event but the occurrence does not complete within the time frame specified, UEM will set that occurrence to a `rejected` state.
- If a UEM Server does not detect any occurrence of a defined system event within the allotted time, UEM will set the entire event to an `expired` state.

UEM Server may follow up each of these states' responses with the execution of a system command or script specified by what is known as an event handler. An event handler's parameters can come from the command options specified for a UEM Manager or from a stored record in the event handler database.

For each event monitored, up to three different event handlers can be specified, one for each of the states (`triggered`, `rejected`, `expired`) described above.

Finally, a UEM Server is used to validate database maintenance requests issued by the UEMLoad utility.

UEMLoad is a client application that serves as the user interface for maintaining records in the event definition and event handler databases. Record parameters are passed to UEMLoad via its available command options. These parameters then are forwarded to a local UEM Server, which validates the parameters and supplies configured default values for any required parameters that were omitted in the request. The UEM Server then forwards the request to the Universal Broker so it actually can be applied to the appropriate database.

# 7.2  Demand-Driven vs. Event-Driven

References are made throughout this manual to *demand-driven* and *event-driven* UEM Servers. Both types of UEM Servers execute from the same application file, `uemsrv.exe`, (`uemsrv` on UNIX), and both types monitor events in essentially the same way.

However, the factors that distinguish a demand-driven Server from an event-driven Server include the following:

- Method in which the UEM Server component is defined
- Method used to initiate the startup of the UEM Server process
- Ability to recover previously monitored events during startup
- Method used to provide the UEM Server with event definitions and event handlers
- Destination of output for event monitoring activity
- Ability to refresh an active UEM Server's configuration
- Duration of the UEM Server process

## 7.2.1  Component Definitions

The values of certain parameters in the component definitions for a demand-driven UEM Server and event-driven UEM Server is one way to distinguish between them.

One parameter in particular, `component_type`, always can be relied upon to identify a demand-driven and an event-driven UEM Server:

- Demand-driven Server has a component type of `uemd`.
- Event-driven Server's component type is `uems`.

A demand-driven UEM Server also relies upon a command line argument, `-demand yes`, to know whether it should run in demand-driven or event-driven mode. This argument is supplied to the UEM Server process via the `start_command` parameter of the component definition. This value is set at installation time, and should not be changed.

## 7.2.2  Server Startup

As a Universal Products component, both types of UEM Server only can be started by a Universal Broker.

A demand-driven UEM Server is started by Universal Broker when the Universal Broker receives a request to do so from one of the UEM client applications (that is, the UEM Manager or UEMLoad). Since a user request drives the startup of the UEM Server process, the process is said to be executed on demand; hence the term demand-driven Server.

An event-driven UEM Server typically is started automatically during Universal Broker startup. A UEM client application is not required to start an event-driven Server.

The Universal Control utility can be used to initiate the startup of an event-driven UEM Server. While it may appear that Universal Control is starting the UEM Server, it merely is, in fact, providing an interface from which the start request can be sent to the Universal Broker. The Universal Broker still is solely responsible for starting the UEM Server.

For complete information on Universal Control, see the Universal Products Utilities 3.2.0 User Guide.

## 7.2.3  Event Recovery

All event monitoring activity conducted by a demand-driven or event-driven UEM Server is recorded in the event spool database. An event spool record is created when an occurrence of an event is detected, or when an event expires. This record also is used to track the status of any event handlers that are executed for the event.

When a demand-driven UEM Server ends, no association is maintained between the Server component and the event spool records it created. If an event occurrence is being tracked when a demand-driven Server ends, the processing state of the record is set to `REJECTED`  (or `CANCELLED` if the UEM Manager was terminated forcefully). If a handler is being executed for an event or event occurrence, the handler status is set to `SHUTDOWN`. Unless the spool records are being retained by the Universal Broker (see the Stoneman's Tip, below), they are deleted when the UEM Server process ends.

When an event-driven UEM Server ends, any event spool records with a processing state of `TRACKING` or `EXECUTING` are retained automatically. Event spool records with all other processing states are only retained under the condition described in the Stoneman's Tip, below.

When the event-driven UEM Server is re-started, it will retrieve (based on component name) all event spool records that it retained from its previous run. If it finds any event occurrences whose processing state is `TRACKING`, it will resume tracking that occurrence. If the record indicates that an event handler process was `EXECUTING` when the UEM Server was brought down, that record's handler status will be set to `UNRECOVERABLE`. This is necessary because the only way to retrieve the handler process's status is via a process ID that could very likely no longer exist in the system. Further, if the process ID does exist, there is no way of knowing whether it belongs to the same process recorded in the event spool record.

> An option can be set in the Universal Broker's configuration to prevent it from deleting any event spool records when the UEM Server component completes. Setting the `comp_info_retention` option to a value greater than 0 causes the event spool record to be preserved.
>
> Because there is currently no database cleanup routine available, this option should be set only following a recommendation from, and with the assistance of, Stonebranch Inc. Customer Support.

**Stoneman's Tip**

Currently, event spool records only can be viewed with the Universal Spool List utility, `uslist`.

Figure 7.1, below, illustrates the syntax used for viewing event spool records.

```
uslist -list uems [-id recid]
```

Figure 7.1  USList Command Syntax for Viewing Event Spool Records

An optional event spool record ID, specified by `recid`, can be provided to obtain complete detail for a singe record.

Figure 7.2, below, illustrates a sample summary and detailed output listing of event spool records.

```
> uslist -list uems
SERIAL NO EVENT ID PRC STATE  HANDLER ID EXIT CODE  EXIT STATUS HANDLER STATUS
1         uemsam1  Triggered  uemsam_t1  0          Normal      Successful
2         uemsam2  Triggered  uemsam_t2  0          Normal      Successful
3         uemsam3  Expired    uemsam_x3  0          Normal      Successful
4         uemsam2  Tracking   N/A        0          Default     Default
5         uemsam1  Triggered  uemsam_t1  0          Default     Executing


> uslist -list uems -id 2
Serial No...............: 2
Event ID................: uemsam2
Component Name..........: uems
Component Description...: Universal Event Monitor Server
Component Version.......: 3.2.0 Level 0 Release Build 109
Component ID............: 1116252422
Event Type..............: FILE
System Object...........: C:\UEMFiles\uemsam2.dat
Processing State........: Triggered
Handler ID..............: uemsam_t2
User Cmd................: N/A
Process ID..............: 2572
User ID.................: sparkie
Start Time..............: 5/16/2005 10:08:52 AM
End Time................: 5/16/2005 10:08:55 AM
Exit Code...............: 0
Exit Status.............: Normal
Handler Status..........: Successful
Last Modified On........: 5/16/2005 10:08:55 AM
```

Figure 7.2  Sample Event Spool Listings

For additional information on all utilities provided with Universal Products, see the Universal Products Utilities 3.2.0 User Guide.

# 7.2.4  Source of Event Parameters

A demand-driven UEM Server relies upon its associated UEM Manager for event definition and event handler information. All the information needed to define the event and describe event handlers come from the command options used to invoke the UEM Manager. While these command options may reference the ID of an event definition or event handler stored in their respective database, the existence of such a record is not required.

Conversely, an event-driven UEM Server relies completely upon these stored event definition and event handler records for its input. One of the first things an event-driven UEM Server component does upon startup is retrieve a list of event definitions that are assigned to it. It then determines which events are active, and begins monitoring them.

# 7.2.5  Reporting of Monitoring Activity

Demand-driven and event-driven UEM Servers record all significant monitoring activity in the Universal Broker log.

This activity includes:

- Detection of event occurrences
- State changes for events and event occurrences
- Results of event handler execution
- Recovery of outstanding event occurrences during process startup

This activity only is viewable on the system where the UEM Server executes. To allow monitoring activity to be followed from the system where the monitoring request was issued, a demand-driven UEM Server also provides feedback to a waiting UEM Manager.

Monitoring activity for demand-driven and event-driven UEM Servers is also captured in the event spool database.

See the Section 7.2.3 Event Recovery and Section 2.6.5 Event Spool Database for more information on the event spool database.

# 7.2.6  Configuration Refresh

Because an event-driven UEM Server typically is a long-running process, the ability to refresh an active UEM Server's configuration and list of assigned event definitions is provided. Automatic refresh of configuration and event information for a demand-driven UEM Server is not supported; the values it obtains at startup are the ones it uses throughout its lifetime.

When a change is made to the stored UEM Server configuration settings (see Section 2.2.5 Configuration File), active event-driven UEM Servers must be notified that a change has taken place. This is done with Universal Control, using the REFRESH_CMD option, along with a component type value that identifies the component to refresh.

Figure 7.3, below, illustrates the command syntax that demonstrates how to use Universal Control to refresh an event-driven UEM Server's configuration.

For more information on Universal Control, see the Universal Products Utilities 3.2.0 User Guide.

```
uctl -host myhost -refresh uems
```

Figure 7.3  Configuration Refresh using Universal Control

**Windows**

A request to update the configuration of local event-driven UEM Servers is issued automatically whenever a change is made to a UEM Server's configuration through the Universal Configuration Manager (see Section 2.4 Universal Configuration Manager).

When Universal Control or the Universal Configuration Manager (Windows only) instructs an active event-driven UEM Server to refresh its cached configuration, the event-driven Server processes the request immediately.

The UEMLoad utility automatically notifies an event-driven UEM Server of an event definition change via a flag that resides in the local Universal Broker. UEM Server checks this flag every two minutes and updates its cached list of event definitions whenever UEMLoad updates them. This eliminates the need to refresh UEM Server with Universal Control following a database change.

# 7.2.7  Process Lifetime

A demand-driven Server is responsible only for monitoring a single event. When that event's inactive date/time has elapsed, or the maximum number of event occurrences has been detected, the event will be made inactive and the UEM Server process will end.

Conversely, an event-driven UEM Server executes until it receives a shutdown request from the Universal Broker that started it. This shutdown request typically is sent during Universal Broker shutdown, when it forces the termination of all active components.

Universal Control can be used to initiate the shutdown of an event-driven UEM Server. While it may appear that Universal Control is stopping the UEM Server component, it is, in fact, merely providing an interface from which the stop request may be sent to the Universal Broker. The UEM Server will stop only after it receives the shutdown request from the Universal Broker.

For complete information on Universal Control, see the Universal Products Utilities 3.2.0 User Guide.

# Chapter 8
# Universal Event Monitor Server for Windows

## 8.1 Overview

This chapter describes the information for running the Universal Event Monitor Server under the Windows operating system.

This information is separated into the following sections:

- Server Environment
- Component Definition
- Configuration
- Security

# 8.2 Server Environment

UEM Server is a Universal Broker-managed component, which means it only can be started and (gracefully) shut down by the Universal Broker. It runs as a background process and, except for displaying version and copyright information, it provides no native user interface or console interaction.

## Execution Context

The security context that a UEM Server executes with depends on UEM Server configuration options and whether the Server is demand-driven or event-driven (see Section 7.2 Demand-Driven vs. Event-Driven).

If the USER_SECURITY option is enabled in the UEM Server configuration, a demand-driven UEM Server executes in the security context of the user account specified via the UEM Manager. If security is not enabled in the UEM Server configuration, it runs in the security context of the Universal Broker.

An event-driven UEM Server always executes in the security context of the Universal Broker, regardless of the value of the security configuration option.

# 8.3 Component Definition

All Universal Products components managed by Universal Broker have a component definition. The component definition is a text file of options containing component-specific information required by Universal Broker.  (For details on how Universal Broker manages components, see the Universal Broker 3.2.0 User Guide.)

The syntax of a component definition file is the same as a configuration file. See Section 2.2.6 Configuration File Syntax for detailed syntax information.

Although component definition files can be edited with any text editor (for example, Notepad), the Universal Configuration Manager application is the recommended way to edit component definitions for Windows (see Section 2.4 Universal Configuration Manager).

Note:   The component definitions for all Universal Products are identified in the Component Definitions property page of the Universal Broker (see Figure 8.1, below).
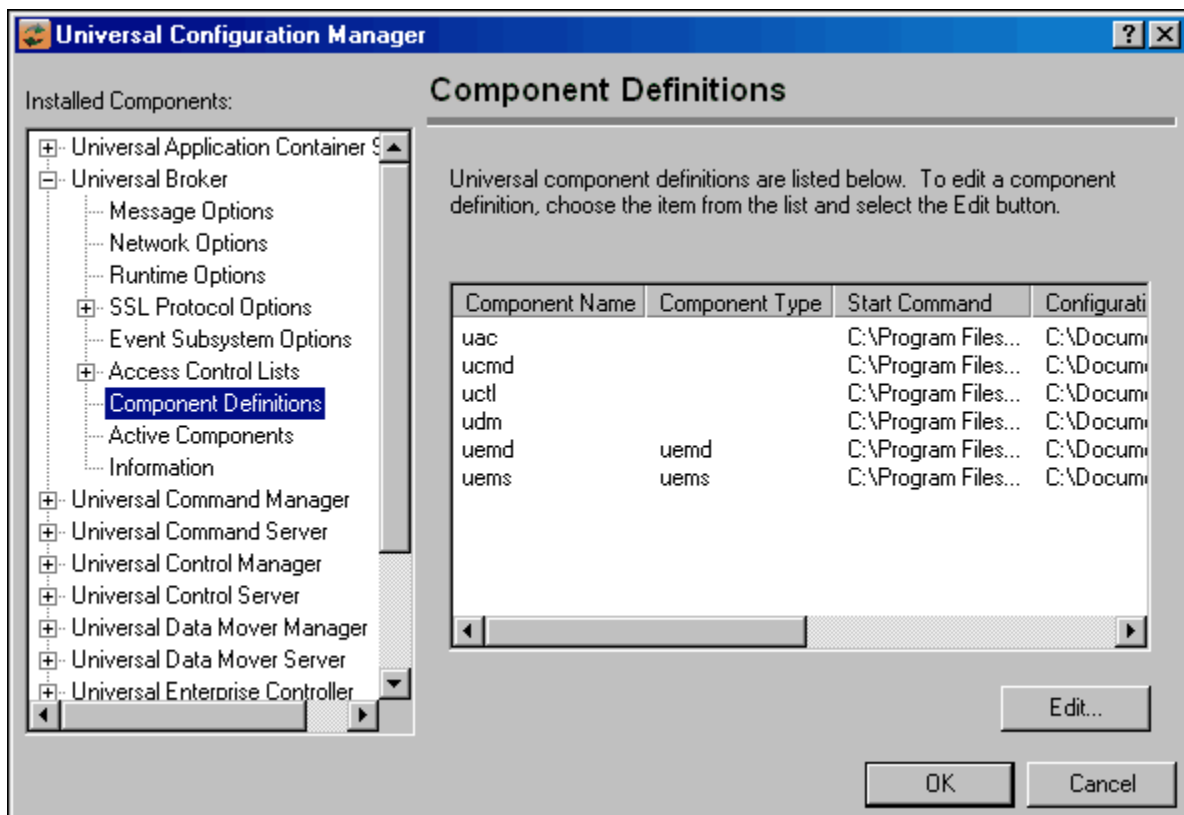


Figure 8.1  Universal Configuration Manager - Component Definitions

Table 8.1 identifies all of the options that comprise the UEM Server for Windows component definition.

Each **Option Name** is a link to detailed information about that component definition option in the Universal Event Monitor 3.2.0 Reference Guide.

| Option Name | Description |
|---|---|
| AUTOMATICALLY_START | Specification for whether or not UEM Server starts automatically when Universal Broker is started |
| COMPONENT_NAME | Name by which the clients know the UEM Server |
| COMPONENT_TYPE | Type of UEM Server (demand-driven or event-driven) |
| CONFIGURATION_FILE | Full path name of the UEM Server configuration file |
| RUNNING_MAXIMUM | Maximum number of UEM Servers that can run simultaneously |
| START_COMMAND | Full path name of the UEM Server program |
| WORKING_DIRECTORY | Full path name of the UEM Server working directory |

Table 8.1  UEM Server for Windows - Component Definition Options

# 8.4  Configuration

All UEM Servers defined on a particular system share a single configuration. A demand-driven UEM Server reads configuration options when it starts, and uses those values while it executes. Because an event-driven UEM Server can run for an extended period of time, the configuration information it caches at start-up can be refreshed whenever the stored configuration options are refreshed (see Section 7.2.6 Configuration Refresh).

A UEM Server's configuration consists of default values that are used to establish the runtime environment and provide missing values for required event definition and event handler parameters.

For a detailed discussion on configuration methods for all Universal Products, see Section 2.2.1 Configuration Methods.

The UEMLoad utility can override a configured default via its command options when it adds records to the event definition and event handler databases. Also, a UEM Manager can override options for a demand-driven UEM Server when it submits a monitoring request.

The syntax for those overrides, when available, are included with the description of those options.

# 8.4.1  Configuration Options

Table 8.2, below, categorize available UEM Server configuration options according to function.

| Category | Description |
|---|---|
| Event Definition | Allow default values to be set for any required event definition parameters that were omitted from the UEM Manager or UEMLoad utility request. |
| Event Handler | Allow default values to be set for any required event handler parameters that were omitted from the UEM Manager or UEMLoad utility request. |
| Installation | Options that specify installation requirements, such as directory locations |
| Message | UEM message options. |
| Monitoring | Control UEM activity. |
| Network | Manage data transmission between local and remote systems. |

Table 8.2  UEM Server for Windows - Configuration Options Categories

The UEM Server configuration options for each of the categories listed in Table 8.2 are summarized in the following tables. Each Option Name is a link to detailed information about that option in the Universal Event Monitor 3.2.0 Reference Guide.

## Event Definition Category Options

| Option Name | Description |
|---|---|
| ASSIGNED_COMPONENT_NAME | UEM Server component to which a stored event definition is assigned. |
| TRACKING_INTERVAL | Frequency, in seconds, with which a tracked event occurrence is tested for completeness. |

### Type-Specific Event Definition Options
These options are specific to event definitions with an event type of `FILE`.

| Option Name | Description |
|---|---|
| MINIMUM_FILE_SIZE | Smallest size a file may be in order for it to be considered complete. |
| RENAME_FILE | Flag that indicates whether or not a completed file should be renamed. |
| RENAME_FILE_SPECIFICATION | Name or pattern to use when a file is renamed. |

# Event Handler Category Options

| Option Name | Description |
| --- | --- |
| EXPIRED_HANDLER_ID | ID of a stored event handler record that should be used whenever an event is set to an expired state. |
| INTERACT_WITH_DESKTOP | Specifies whether or not the desktop of the current interactive logon session is accessible to the handler process. |
| LOGIN | Specifies whether or not to load a user's profile and environment when a handler process is executed. |
| LOGON_METHOD | Specifies how a user account is logged on when the environment for execution of a handler process is established. |
| MAXIMUM_RETURN_CODE | Highest return code than an event handler may exit with to still be considered as having executed successfully. |
| REJECTED_HANDLER_ID | ID of a stored event handler record that should be used whenever an event occurrence is set to a rejected state. |
| TRIGGERED_HANDLER_ID | ID of a stored event handler record that should be used whenever an event occurrence is set to a triggered state. |
| USER_SCRIPT_TYPE | Type of script interpreter used to evaluate and execute a script on behalf of an event handler. |
| USER_SECURITY | Specifies whether or not user account authorization is required to monitor an event and/or execute a handler process. |

# Installation Category Options

| Option Name | Description |
| --- | --- |
| INSTALLATION_DIRECTORY | Base directory in which the product is installed |

# Message Category Options

| Option Name | Description |
| --- | --- |
| MESSAGE_LEVEL | Level of messages printed. |
| NLS_ DIRECTORY | Directory location of message catalog and code page tables |
| TRACE_DIRECTORY | Specifies the directory in which trace files are written. |
| TRACE_FILE_LINES | Maximum number of lines written to a trace file before it wraps around. |
| TRACE_TABLE | Memory trace table specification. |

## Monitoring Category Options

| Option Name | Description |
|---|---|
| POLLING_INTERVAL | Frequency with which a UEM Server will detect new occurrences of a system event. The UEM Server will also check at this time see if a monitored event should be made inactive. |

## Network Category Options

| Option Name | Description |
|---|---|
| CODE_PAGE | Code page used for text translation. |

# 8.5  Security

UEM Server for Windows is designed to be a secure system. As the level of security rises, so does the administrative complexity of the system. UEM Server has balanced the two to avoid the administrative complexity with a minimum sacrifice to security.

Security issues addressed by UEM Server include:

1.  Access to UEM Server files and directories
2.  Access to UEM Server configuration options
3.  UEM Server user account
4.  Privacy and integrity of transmitted network data
5.  Access to UEM Server features

## 8.5.1  Data Privacy

Data transmitted to a UEM Server across a network connection is protected using features present in all Stonebranch Inc. Universal Products.

For more information on the steps taken to protected transferred data, see Section 2.5.1 Security.

## 8.5.2  File Permissions

Only trusted user accounts, which are most likely those that are members of the Administrators group, should have write access to the UEM Server installation directory and subdirectories, and the files within them. Authorized users of UEM require read access to the message catalogs (`*.umc` files), which reside in the `.\Universal\nls` directory. If UEM Server is installed on an NTFS partition, these file permissions are automatically set during installation.

The component definitions for demand-driven and event-driven UEM Servers include the location of a WORKING_DIRECTORY. By default, this is `.\Universal\UEMHome`.

When the USER_SECURITY option is enabled, and before a demand-driven UEM Server begins monitoring an event or an event-driven UEM Server executes an event handler process, the UEM Server will create a subdirectory (if it does not already exist) for the authenticated user under this working directory. The name of the directory matches the ID of the user account specified from the UEM Manager command line or stored in the event handler record. If a Windows domain account is used, the name of the directory is `userid.domain`, where `userid` is the user ID and `domain` is the domain name. After the directory is created, the specified user account is given ownership of it and granted full control over it.

## 8.5.3  Configuration Options

Configuration options are set by the Universal Configuration Manager, which only can be executed by accounts that are members of the Administrators group (see Section 2.4 Universal Configuration Manager).

## 8.5.4  User Authentication

When the USER_SECURITY option is enabled, a demand-driven UEM Server requires the ID and password of a valid local user account before it will begin monitoring the event. Likewise, an event-driven UEM Server requires this information to be stored in an event handler record before it can execute a process on behalf of that handler. All handler processes started by UEM Server when the USER_SECURITY option is enabled are executed in the security context of this user account.

To allow Windows to verify the user account information, a UEM Server will attempt to log that user on to the system via a call to a Windows system function.

Windows provides two types of logon methods: interactive and batch. Unless they have been explicitly denied the ability to do so, most user accounts can be validated with the interactive logon method. Conversely, a user account typically must be granted an additional privilege before they can be authenticated using the batch logon method. This privilege is shown in Windows as "Log on as a batch job."

For information on configuring UEM Server to use this logon method, see the LOGON_METHOD configuration option in the Universal Event Monitor 3.2.0 Reference Guide.

# 8.5.5  Universal Access Control List

The UEM Server uses Universal Access Control Lists (UACLs) as an extra layer of security. UEM Server uses ACLs to grant or deny access to its own execution, execution of event handler processes, and database maintenance.

For an overview on Universal Access Control List features, see Section 2.7 Universal Access Control List.

## UACL Entries

The syntax of a UACL entry file is the same as the Universal Event Monitor configuration file. See Section 2.2.6 Configuration File Syntax for detailed syntax information.

Table 8.3, below, identifies all Universal Event Monitor for Windows UACL entries. Each **UACL Entry Name** is a link to detailed information about that UACL entry in the Universal Event Monitor 3.2.0 Reference Guide.

| UACL Entry Name | Description |
|---|---|
| ACCESS_ACL | Allows or denies access to a demand-driven Universal Event Monitor Server. |
| DATABASE_MAINTENANCE_ACL | Allows or denies a user account access to the event definition and event handler databases. |
| EVENT_HANDLER_ACL | Allows or denies a user account the authority to execute an event handler process. |

Table 8.3  Universal Event Monitor Server for Windows – UACL Entries

## Updating the Universal Event Monitor Server ACL Entries

Although UACL files can be edited with any text editor (for example, Notepad), the Universal Configuration Manager application, accessible via the Control Panel, is the recommended way to update UACL entries (see Section 2.4 Universal Configuration Manager). From there, ACL entries can be added, changed, deleted, or sorted (rules are applied in the order in which they are listed).

Figure 8.2, below, illustrates an example. The set of ACL entries only allows connections from host 10.20.30.40 if the user on that host is TS1004. All other remote users will be blocked. TS1004 may run processes on the local system using any user account, provided the correct password is supplied. No processes may be run with Universal Event Monitor using the Administrator account on the local system, regardless of where the request originated.
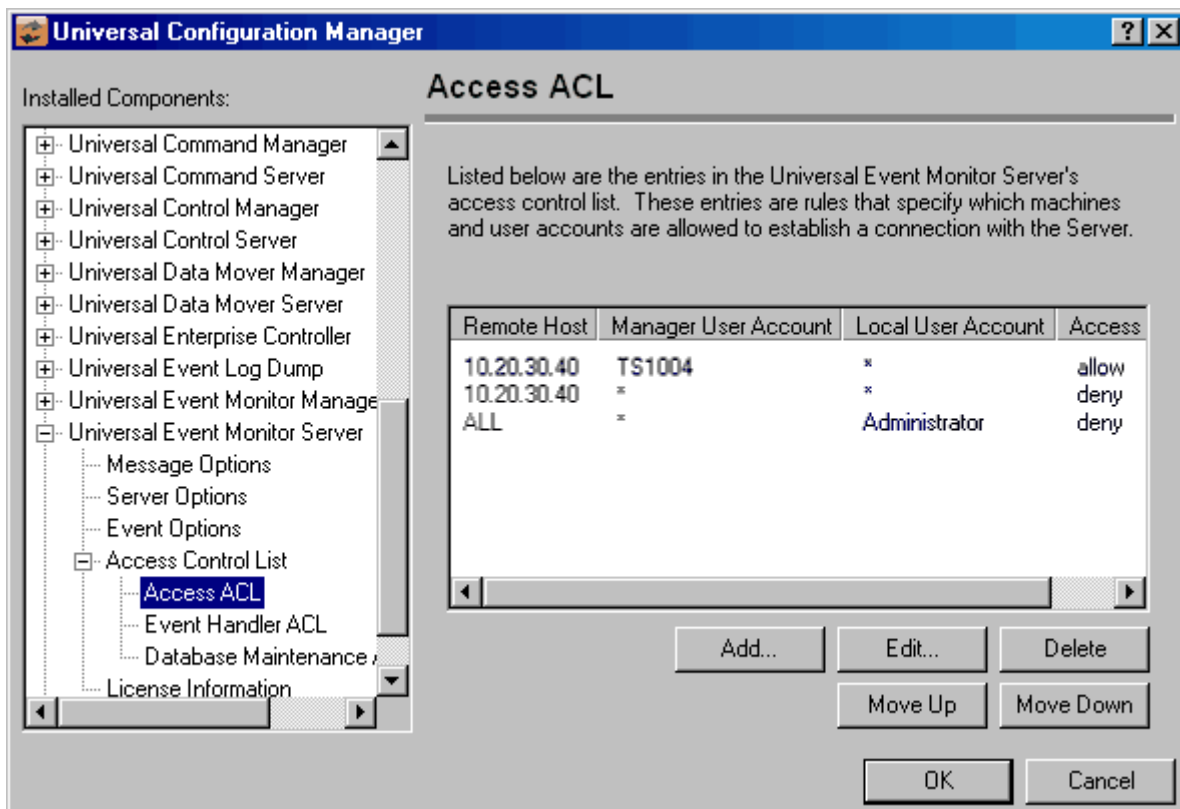


Figure 8.2  Universal Configuration Manager - Universal Event Monitor Server - Access ACL

# Chapter 9
# Universal Event Monitor
# Server for UNIX

## 9.1 Overview

This chapter describes the information for running the Universal Event Monitor Server under the UNIX operating system.

This information is separated into the following sections:

- Server Environment
- Component Definition
- Configuration
- Security

# 9.2  Server Environment

UEM Server is a Universal Broker-managed component, which means it only can be started and (gracefully) shutdown by the Universal Broker. It runs as a background process and, except for displaying version and copyright information, it provides no native user interface or console interaction.

# 9.2.1  Execution Context

The security context that a UEM Server executes with depends on UEM Server configuration options and whether the Server is demand-driven or event-driven (see Section 7.2 Demand-Driven vs. Event-Driven).

If the USER_SECURITY option is enabled in the UEM Server configuration, a demand-driven Server executes in the security context of the user account specified via the UEM Manager's command options.

If security is not enabled in the UEM Server configuration, it runs in the security context of the Universal Broker.

An event-driven UEM Server always executes in the security context of the Universal Broker, regardless of the value of the security configuration option.

# 9.3  Component Definition

All Universal Products components managed by Universal Broker have a component definition. The component definition is a text file of options containing component-specific information required by Universal Broker. (For details on how Universal Broker manages components, see the Universal Broker 3.2.0 User Guide.)

The syntax of a component definition file is the same as a configuration file. See Section 2.2.6 Configuration File Syntax for detailed syntax information.

The UEM Server for UNIX component definition is located in the `/etc/universal/comp` directory.

Table 9.1, below, identifies all of the options that comprise the UEM Server for UNIX component definition.

Each **Option Name** is a link to detailed information about that component definition option in the Universal Event Monitor 3.2.0 Reference Guide.

| Option Name | Description |
| --- | --- |
| AUTOMATICALLY_START | Specification for whether or not UEM Server is started automatically when Universal Broker is started |
| COMPONENT_NAME | Name by which clients know the UEM Server |
| COMPONENT_TYPE | Type of UEM Server (demand-driven or event-driven) |
| CONFIGURATION_FILE | Full path name of the UEM Server configuration file |
| RUNNING_MAXIMUM | Maximum number of UEM Servers that can run simultaneously |
| START_COMMAND | Full path name of the UEM Server program |
| WORKING_DIRECTORY | Full path name of the UEM Server working directory |

Table 9.1  UEM Server for UNIX - Component Definition Options

# 9.4  Configuration

All UEM Servers defined on a particular system share a single configuration. A demand-driven UEM Server reads configuration options when it starts, and uses those values while it executes. Because an event-driven Server can run for an extended period of time, the configuration information it caches at start-up can be refreshed whenever the stored configuration options are refreshed (see Section 7.2.6 Configuration Refresh).

A UEM Server's configuration consists of default values that are used to establish the runtime environment and provide missing values for required event definition and event handler parameters.

For a detailed discussion on configuration methods for all Universal Products, see Section 2.2.1 Configuration Methods.

Stored configuration options reside in the UEM Server configuration file. The name of this file can be set in the UEM component definition file. The default name for this file is `uems.conf`. This is a plain text file that can be edited with any text editor.

Configuration options stored in the configuration file are done so using a keyword/value combination. The keywords for each configuration option are included along with the description of the option.

The UEMLoad utility can override a configured default via its command options when it adds records to the event definition and event handler databases. Also, a UEM Manager can override options for a demand-driven UEM Server when it submits a monitoring request. The syntax for those overrides, when available, are included with the option's description.

# 9.4.1  Configuration Options

Table 9.2, below, categorizes available UEM Server configuration options according to function.

| Category | Description |
|----------|-------------|
| Event Definition | Allow default values to be set for any required event definition parameters that were omitted from the UEM Manager or UEMLoad utility request. |
| Event Handler | Allow default values to be set for any required event handler parameters that were omitted from the UEM Manager or UEMLoad utility request. |
| Installation | Options that specify installation requirements, such as directory locations |
| Message | UEM message options. |
| Monitoring | Control UEM activity. |
| Network | Manage data transmission between local and remote systems. |

Table 9.2  UEM Server for UNIX - Configuration Options Categories

The UEM Server configuration options for each of the categories listed in Table 9.2 are summarized in the following tables. Each Option Name is a link to detailed information about that option in the Universal Event Monitor 3.2.0 Reference Guide.

## Event Definition Category Options

| Option Name | Description |
|-------------|-------------|
| ASSIGNED_COMPONENT_NAME | UEM Server component to which a stored event definition is assigned. |
| TRACKING_INTERVAL | Frequency, in seconds, with which a tracked event occurrence is tested for completeness. |

### Event Definition Options - Type-Specific
These options are specific to event definitions with an event type of `FILE`.

| Option Name | Description |
|-------------|-------------|
| MINIMUM_FILE_SIZE | Smallest size a file may be in order for it to be considered complete. |
| RENAME_FILE | Flag that indicates whether or not a completed file should be renamed. |
| RENAME_FILE_SPECIFICATION | Name or pattern to use when a file is renamed. |

## Event Handler Category Options

| Option Name | Description |
|---|---|
| EXPIRED_HANDLER_ID | ID of a stored event handler record that should be used whenever an event is set to an expired state. |
| LOGIN | Specifies whether or not to load a user's profile and environment when a handler process is executed. |
| MAXIMUM_RETURN_CODE | Highest return code than an event handler may exit with to still be considered as having executed successfully. |
| REJECTED_HANDLER_ID | ID of a stored event handler record that should be used whenever an event occurrence is set to a rejected state. |
| SHELL | Default shell interpreter for event handler processes executed as scripts. |
| TRIGGERED_HANDLER_ID | ID of a stored event handler record that should be used whenever an event occurrence is set to a triggered state. |
| USER_SECURITY | Specifies whether or not user account authorization is required to monitor an event and/or execute a handler process. |

## Installation Category Options

| Option Name | Description |
|---|---|
| INSTALLATION_DIRECTORY | Base directory in which the product is installed |

## Message Category Options

| Option Name | Description |
|---|---|
| MESSAGE_LEVEL | Level of messages printed. |
| NLS_ DIRECTORY | Directory location of message catalog and code page tables |
| TRACE_DIRECTORY | Directory where trace files are written. |
| TRACE_FILE_LINES | Maximum number of lines written to a trace file before it wraps around. |
| TRACE_TABLE | Memory trace table specification. |

## Monitoring Category Options

| Option Name | Description |
|---|---|
| POLLING_INTERVAL | Frequency with which a UEM Server will detect new occurrences of a system event. The UEM Server also will check at this time see if a monitored event should be made inactive. |

## Network Category Options

| Option Name | Description |
|---|---|
| CODE_PAGE | Code page used for text translation. |

# 9.5  Security

UEM Server for Unix is designed to be a secure system. As the level of security rises, so does the administrative complexity of the system. UEM Server has balanced the two to avoid the administrative complexity with a minimum sacrifice to security.

Security issues addressed by UEM Server include:

1. Access to UEM Server files and directories
2. Access to UEM Server configuration files
3. UEM Server user account
4. Privacy and integrity of transmitted network data
5. Access to UEM Server features

## 9.5.1  Data Privacy

Data transmitted to a UEM Server across a network connection is protected using features present in all Stonebranch Inc. Universal Products.

For more information on the steps taken to protected transferred data, see Section 2.5.1 Security.

## 9.5.2  File Permissions

Only trusted user accounts should have write access to the UEM Server installation directory and subdirectories, and the files within them. Authorized users of UEM require read access to the message catalogs (`*.umc` files), which reside in the `./universal/nls` directory.

## 9.5.3  Configuration Files

Only trusted user accounts should have write access to the UEM Server's configuration file.

For information on storing configuration options, see Section 2.2.5 Configuration File.

## 9.5.4  User Authentication

When the USER_SECURITY option is enabled, a demand-driven UEM Server requires the ID of a valid local user account before it will begin monitoring the event. A password also may be required, depending on the rules set up in the ACCESS_ACL.

Likewise, an event-driven UEM Server requires this information to be stored in an event handler record before it can execute a process on behalf of that handler. All handler processes started by UEM Server when the USER_SECURITY option is enabled are executed in the security context of this user account.

UEM Server for UNIX supports three different types of user authentication methods:

1.  Default authentication uses the UNIX traditional password comparison method.
2.  PAM authentication uses the PAM API to authenticate users. This option is only available for certain UNIX platforms.
3.  HP-UX Trusted Security uses HP-UX Trust Security APIs to authenticate users. This is only available on Hewlett Packard HP-UX platforms.

## 9.5.5  Universal Access Control List

The UEM Server uses Universal Access Control Lists (UACLs) as an extra layer of security. UEM Server uses ACLs to grant or deny access to its own execution, execution of event handler processes, and database maintenance.

For an overview on Universal Access Control List features, see Section 2.7 Universal Access Control List.

### UACL Entries

The syntax of a UACL entry file is the same as the Universal Event Monitor configuration file. See Section 2.2.6 Configuration File Syntax for detailed syntax information.

Table 9.3, below, identifies all Universal Event Monitor for UNIX UACL entries. Each **UACL Entry Name** is a link to detailed information about that UACL entry in the Universal Event Monitor 3.2.0 Reference Guide.

| UACL Entry Name | Description |
| --- | --- |
| ACCESS_ACL | Allows or denies access to a demand-driven Universal Event Monitor Server. |
| DATABASE_MAINTENANCE_ACL | Allows or denies a user account access to the event definition and event handler databases. |
| EVENT_HANDLER_ACL | Allows or denies a user account the authority to execute an event handler process. |

Table 9.3  Universal Event Monitor Server for UNIX – UACL Entries

# Chapter 10
# Universal Event Monitor
# Load Utility

## 10.1 Overview

A Universal Event Monitor (UEM) Server has three database files that it can use during event processing:

1. `ueme.db` stores event definitions.
2. `uemh.db` stores event handlers.
3. `uems.db` is a spool file that records all activity related to event monitoring.

The UEMLoad utility (`uemload`) manages the event definition and event handler database files. (For information on the spool database file, see Chapter 7 Universal Event Monitor Server.)

UEMLoad can be used to:

- Add, update, and delete event definitions and/or event handlers from their respective database files.
- List the entire contents of the event definition and/or event handler database files.
- List the parameters of a single event definition and/or event handler.
- Export the contents of the event definition and/or event handler database files to a file that can be used to re-initialize the database or populate a new database on another system.

By design, UEMLoad itself only can access local event definition and event handler database files. However, it is possible to store definition load files in a single location (for example, a PDS on a z/OS system) and centrally manage their distribution to remote systems using Universal Command.

When a definition load file is redirected from `stdin` to Universal Command, Universal Command will in turn forward the redirected `stdin` to a remote instance of UEMLoad. UEMLoad then behaves as though it were reading a local definition load file.

# 10.2  Database Files

This section describes the contents of the following database files:

- Event definition
- Event handler

For operating system-specific information on the UEMLoad utility, see Chapter 11 UEMLoad Utility for Windows and Chapter 12 UEMLoad Utility for UNIX.

# 10.2.1  Database Files Location

The event definition and event handler database files are local to each system. The files are created automatically when the initial attempt to open them is made. They reside in a directory that, if it does not already exist, is created during product installation.

**UNIX**

The default database directory is `/var/opt/universal/spool`.

**Windows**

The default database directory is `C:\Program Files\Universal\spool`.

The event definition and event handler database files reside in a subdirectory named `ubroker`. This subdirectory is used only on Windows installations to improve database performance. The name chosen for the directory signifies that all access to the event definition and event handler databases is routed through Universal Broker.

Additional installation recommendations and requirements for all database files used by Universal Products components can be found in the Universal Products 3.2.0 Installation Guide.

# 10.3  Event Definition Database File

Event definitions are stored in the database file `ueme.db`.

This section describes the parameters that comprise a record in the event definition file.

## 10.3.1  Event Definition Parameters

For UEM, an event definition represents a system event.

The parameters in an event definition record:

- Describe a system event.
- Establish the criteria that UEM uses to test for the completion of an event occurrence.

An event definition record also can contain the ID of a record in the event handler database, which dictates how an event or an event occurrence is handled (that is, responded to) when certain conditions are met.

### Event Definition Parameters - Categories

Event definition parameters fall into two categories:

1. Parameters that are present in all event definition records. The value of one of these general parameters, Event Type, dictates the parameters that fall under the second category.
2. Parameters that are specific to an Event Type. Currently, UEM supports a single event type, `FILE`, which detects the creation of a file.

# 10.3.2  Event Definition Parameters - General

Table 10.1, below, describes the general event definition parameters.

The parameters that make up the record's key are underlined.

| Parameter | Description | Remarks |
|---|---|---|
| Event ID | Unique identifier for the event definition. | Length of the handler ID cannot be greater than 32 characters.<br><br>This value is case-sensitive. |
| Assigned UEM Component | Name of an event-driven Universal Event Monitor Server component responsible for monitoring the event. | |
| Enabled Flag | A *true* / *false* value that determines whether an event-driven UEM Server processes the event definition. | If *true*, the event is checked periodically by its assigned event-driven UEM Server to see if it should be made active or inactive.<br><br>If *false*, the event, while still included in a UEM Server's list of assigned components, is not checked to see if it should be made active.<br><br>[Default is *true*.] |
| Active Flag | A *yes / no* value that indicates whether UEM Server is monitoring the event. | If *yes*, the current date/time is greater than the defined **Activation Date/Time**, but less than the defined **Inactivation Date/Time**.<br><br>If *no*, the current date/time is greater than the defined **Inactivation Date/Time**.<br><br>This parameter is set only through UEM Server. It cannot be set using the UEMLoad utility.<br><br>[Default is *no*.] |
| Event Type | System event represented by the event definition. | Value corresponds to one of the supported event types.<br><br>(Currently, `FILE` is the only support event type.) |
| Activation Date | Date on which UEM will begin checking for the occurrence of the system event represented by the event definition. | Specified using the format *YYYY.MM.DD*.<br><br>[Default is current date.] |
| Activation Time | Time on the **Activation Date** at which UEM will begin checking for the occurrence of the system event represented by the event definition. | Specified using the format *HH:MM*.<br><br>[Default is current time.] |
| Inactivation Date | Date on which UEM will stop checking for the occurrence of the system event represented by the event definition. | Specified using the format *YYYY.MM.DD*.<br><br>[Default is *2038.01.16* for event definitions stored with the UEMLoad utility. |
| Inactivation Time | Time on the **Inactivation Date** at which UEM will stop checking for the occurrence of the system event represented by the event definition. | Specified using the format *HH:MM*.<br><br>[Default is *23:59*.] |

| Parameter | Description | Remarks |
|---|---|---|
| Tracking Interval | Frequency with which UEM tests for the completion of an occurrence of the system event represented by the event definition. | Value is expressed in seconds. Value depends on the event type. Some system events will be considered complete as soon as their occurrence is detected by UEM. For these events, this parameter should be *0* (zero). [Default is *10*.] |
| Triggered Event Handler ID | ID of an event handler database record whose specified actions should be taken when an event occurrence satisfies its defined completion criteria. | Value is case sensitive. Value can be blank, in which case the default value specified in the UEM Server configuration (empty string for Windows, *NONE* for UNIX) will be used. |
| Rejected Event Handler ID | ID of an event handler database record whose specified actions should be taken if one or more tracked event occurrences fail to complete before the event is made inactive by UEM. | Value is case sensitive. Value can be blank, in which case the default value specified in the UEM Server configuration (empty string for Windows, *NONE* for UNIX) will be used. |
| Expired Event Handler ID | ID of an event handler database record whose specified actions should be taken if an event is made inactive with no occurrence of the system event represented by the event definition being detected by UEM. | Value is case sensitive. Value can be blank, in which case the default value specified in the UEM Server configuration (empty string for Windows, *NONE* for UNIX) will be used. |
| Handler Options | String that contains one or more parameters that UEM adds to the command line it constructs to execute an event handler's specified command or script. | Added to the command line for every process executed for the event handlers referenced by this event definition. This includes those event handlers referenced by the Triggered Event Handler ID, Rejected Event Handler ID, and Expired Event Handler ID fields. When command line options are specified in the event definition (via Handler Options) and in the event handler (via Options), both are used.  However, the event handler's Options value is added first. |
| Last Update Date/Time | Date and time the event definition was created or last modified. | |
| Last Update User | ID of the user account that created or most recently updated the event definition. | |

Table 10.1  Event Definition Parameters - General

## 10.3.3 Event Definition Parameters - Event Type FILE

Table 10.2, below, describes the parameters that are available in event definitions for an Event Type of `FILE`.

| Parameter | Description | Remarks |
|---|---|---|
| File Specification | Complete path of the file to watch. | File specification can contain wildcards:<br>• ? is available to match up to 1 character.<br>• * can be used to match 0 or more characters. |
| Minimum File Size | Smallest size that a file can be in order to be considered complete. | Value of *0* indicates that a file of any size is acceptable.<br>To specify a storage unit for this value, add either of the following after the file size:<br>• (b)ytes<br>• (k)ilobytes<br>• (m)egabytes<br>• (g)igabtyes<br>If a storage unit is not specified, UEM assumes the file size is given in bytes. |
| Rename File | Indicates whether or not UEM Server will rename the file prior to executing the triggered event handler. This will prevent multiple detections of the same file. | Acceptable values are *yes* or *no*.<br>[Default is *yes*.] |

| Parameter | Description | Remarks |
|---|---|---|
| Rename File Specification | Format used by UEM to rename a file. | Complete file name or a file mask that accepts specific well-known variables that will be substituted by UEM Server at run time. When specified, variables must be in the format `$(var)`.<br><br>The following variables are available:<br><br>• `$(compname)` – component name<br>• `$(compid)` - component ID<br>• `$(date)` - current date in the format YYYYMMDD<br>• `$(time)` - current time, n the format HHMMSS<br>• `$(origname)` - original base file name, minus its last extension<br>• `$(origext)` - original file extension<br>• `$(seqnum)` - sequence number that starts at 0 when a UEM Server component is started, and is then incremented by 1 for each file renamed.<br><br>If no path is specified, the file is simply renamed. Otherwise, the file can be renamed, moved from its original location, and placed in the path specified by this parameter.<br><br>If no value is specified, a default value of `$(compname).$(compid).$(date).$(seqnum)`, specified in the UEM Server configuration, will be used. |

Table 10.2  Event Definition Parameters for Events of Type **FILE**

# 10.4  Event Handler Database File

Event handlers are stored in the database file `uemh.db`.

This section describes the parameters that comprise a record in the event handler file.

## 10.4.1  Event Handler Parameters

For UEM, an event handler represents the action that should be taken in response to a monitored event.

An event handler record consists of parameters that:

- Describe the action to take.
- Establish an execution environment for the action.
- Define success or failure criteria for the action's outcome.

UEM Server executes the specified action on behalf of the event handler. Accordingly, the action also can be referred to as the event handler process.

Table 10.3, below, describes the event handler parameters.

The parameters that make up the record's key are underlined.

| Parameter | Description | Remarks |
|-----------|-------------|---------|
| Handler ID | Unique identifier for the event handler. | Length of the handler ID cannot be greater than 32 characters. Value is case-sensitive. |
| Handler Type | Type of action that will be executed. | If the Handler Actions field contains a system command or other command line application, this field has a value of *cmd*. If the Handler Actions field contains a series of system commands that should be executed as a script file, this field has a value of *script*. |
| Script Type | Script interpreter to invoke to execute script statements contained in the Handler Actions parameter. | On Windows-based systems, this value will be used as a file extension for the temporary script constructed by UEM Server in order to execute the script statements specified in the Handler Actions parameter. To insure proper execution of the script, a file association should exist on the target system that specifies the application with which the script should be executed. This parameter is ignored on UNIX-based systems. This parameter is ignored if the Handler Type has a value of *cmd*. |

| Parameter | Description | Remarks |
|---|---|---|
| Maximum Acceptable Return Code | Highest value that the handler process can return to still be considered as having executed successfully. | If the value returned by the action is less than or equal to the value specified by this parameter, UEM will report a successful completion of the handler process. If the value returned by the action is greater than the value specified by this parameter, UEM will report a failure of the handler process. |
| User ID | ID of the user account in whose security context the handler actions will be run. | If the UEM Server processing this record is configured to run without security, the value in this parameter is ignored. |
| Password | Password of the user account specified in the User ID parameter. | If the UEM Server processing this record is configured to run without security, this parameter is ignored. On UNIX-based systems, a UACL entry can be defined that permits handler execution for some user accounts without requiring authentication. In such a situation, this parameter is ignored. This parameter is encrypted prior to being stored in the database. |
| Encrypted File | Complete path to a file encrypted with Universal Encrypt. The userid and password options can be stored in an external file instead of specifying the handler userid and password parameters. UEM reads the contents of this file to obtain the User ID and Password of a user account, which then is used to establish a security context in which the event handler process is executed. If an encryption key was used other than uencrypt's default, that key can be specified with the Encrypted Key parameter. | A Password may not be necessary on UNIX, depending on the existing EVENT_HANDLER_ACL entries. Storing the path to an encrypted file allows the file to be shared between Universal Product applications. This makes it unnecessary to update individual event handler records whenever account information is updated or security requirements change. |
| Encrypted Key | Optional encryption key used to decrypt the encrypted file. | This must be the same key that was used to encrypt the file. |
| Handler Actions | Command or script executed by UEM Server for this handler. | For records with a Handler Type of *cmd*, this parameter contains the syntax required to execute a system command or other command line application. For records with a Handler Type of *script*, this parameter contains one or more system commands that are executed collectively as a batch script. |
| Last Update Date/Time | Date and time that the event handler was created or last modified. | |
| Last Update User | ID of the user account that created or most recently updated the event handler. | |

| Parameter | Description | Remarks |
|---|---|---|
| Options | String with one or more parameters that UEM adds to the command line that it constructs in order to execute a specified command or script. | This parameter serves the same purpose as the event definition's Handler Options parameter, but differs in scope.<br><br>An event definition's Handler Options are applied to every handler process executed on its behalf. An event definition's Handler Options may specify global handler execution options (for example, output redirection) or may customize event handler behavior for that specific event.<br><br>On the other hand, an event handler's Options value is applied every time the handler is executed, regardless of which event definition causes it to execute.<br><br>When command line options are specified in the event definition (via Handler Options) and in the event handler (via Options), both are used. However, the event handler's Options value is added first. |

Table 10.3  Event Handler Parameters

# 10.5  Controlling Database Access

Universal Broker is primarily responsible for providing access to the Universal Products databases.

However, there are utilities provided, including Universal Spool List (`uslist`) and Universal Spool Remove (`uslrm`) that can be used for direct access to these databases. While these utilities should be used only following a recommendation from and with the assistance of Stonebranch, Inc. Customer Support, they are documented in the Universal Products Utilities 3.2.0 User Guide.

To protect the database contents, operating system permissions on the database files themselves should be set so that only accounts with super-user or administrative privileges have access to them.

For more information on the location, names, and contents of the UEM database files, see Section 10.2.1 Database Files Location.

# 10.5.1  Access via UEMLoad Utility

While the contents of UEM databases can be viewed using Universal Spool List, it is recommended that all access be done using the UEMLoad utility.

The ability to remove event definition and event handler records is provided only with UEMLoad. Universal Spool Remove cannot be used to delete records from those databases.

Only UEMLoad can manage event definition and event handler databases that are local to the system on which the UEMLoad resides. To process a request, the UEMLoad sends a message to the Universal Broker running on that system, instructing it to start a demand-driven UEM Server. A control session is established between UEMLoad and the UEM Server, which provides for direct communication between the two processes. It is over this session that UEMLoad sends the database request to the UEM Server, so that supplied values can be validated and defaults can be provided for any values that were omitted. The UEM Server then forwards the request to the Universal Broker for actual application of the changes to the appropriate database.

UEMLoad executes in the security context of the user account that started it. Since it is the Universal Broker that applies changes to the event definition and event handler databases, any user with the authority to execute UEMLoad will effectively have access to secure resources. It is therefore strongly recommended that the privileges on UEMLoad be set such that only those user accounts with super-user or administrative privileges be allowed to execute it.

## 10.5.2  Universal Access Control List

Support for controlling access to the event definition and event handler databases also is provided by UEMLoad.

A type of Universal Access Control List (UACL) is provided in order to grant or deny local user accounts the authority to execute UEMLoad. The type of database access (that is: add, update, delete, list, and export) allowed for each authorized user also can be defined.

A typical set of UACL entries intended to fully secure the event definition and event handler databases would include an entry for each user authorized to execute UEMLoad. Then, the types of database access permitted for each of the users would be set in those entries. Finally, a single UACL entry that denies access to all other accounts would be defined.

Whenever UEMLoad is executed, the entries in the UACL will be checked. If a match cannot be found which indicates that the user account that started UEMLoad has the authority to access the database and perform the requested operation, the application will terminate with an error.

A more thorough overview of the Universal Access Control List feature can be found in Section 2.7 Universal Access Control List.

# Chapter 11
# UEMLoad Utility
# for Windows

## 11.1 Overview

This chapter provides information on Universal Event Monitor (UEM) Load utility, specific to the Windows operating system.

Note:   Because the UEM Server component is the only UEM component that uses the databases managed by UEMLoad, and because the UEM Server is available only on UNIX and Windows, UEMLoad also is available only on those operating systems. The command syntax and general usage of UEMLoad is the same on both operating systems, except where noted.

UEMLoad manages records in the event definition and event handler databases. The records in these databases then are used as input to a local UEM Server.

UEMLoad is capable of processing multiple event definition and/or event handler records when the parameters for those records are supplied using a text load file. If no load file is specified, the parameters for a single event definition and/or event handler record may be specified from the command line.

Although UEMLoad can access only local event definition
and event handler databases, it is possible to store definition load files
in a single location (for example, a PDS on z/OS)
and distribute them to remote systems via Universal Command.

Simply redirect the definition load file from stdin and
have Universal Command execute UEMLoad on the remote system.
UEMLoad will read the redirected input and process it
just as it would a local definition load file.
This simplifies central administration of remote databases,
because definition load files do not have to be stored
(and managed) across several systems.

**Stoneman's Tip**

# 11.2  Usage

The UEMLoad utility, executes as a command line application.

The syntax for invoking UEMLoad from the command line requires the name of the program, `uemload`, followed by a list of configuration options.

This section describes the configuration and configuration options, command input, and command line syntax of UEMLoad for Windows.

Section 11.3 Examples of UEMLoad for Windows provides examples demonstrating the flexibility of UEMLoad.

# 11.2.1  Configuration

Configuration consists of:

- Setting default options and preferences for all executions of UEMLoad.
- Setting options and preferences for a single execution of UEMLoad.

UEMLoad for Windows receives its configuration options from the following sources:

1. Command line
2. Environment variables

The order of precedence is the same as the list above; command line options being the highest and environment variables being the lowest. That is, options specified via a command line override options specified via environment variables.

See Section 2.2.1 Configuration Methods for complete details on configuration methods and command input for Universal Products.

## Definition and Event Handler Parameters

The configuration options source list, above, applies only to those general configuration options used to control execution of UEMLoad.

For event definition and/or event handler parameters, input can come from one of three mutually exclusive sources:

1. Command line
2. Definition load file
3. Definition load file redirected from `stdin`

## Definition Load File

When record parameters are provided from the command line, only one event definition and/or event handler can be specified. To add, update, or delete multiple event definition and/or event handler records at the same time, use a definition load file.

If a definition load file is specified from the command line (with the **-deffile** option) along with other event definition and/or event handler record parameters (for example, **-event_id** and **-handler_id**), the UEMLoad request will fail.

In this situation, either:

*   Add the event definition and/or event handler parameters to the load file.
*   Remove the **-deffile** option from the request.

If no record parameters are specified from a definition load file or from the command line, UEMLoad assumes a definition load file is provided via standard input (that is, **stdin**) redirection.

If UEMLoad detects no input at all – either from the command line or a definition load file stored locally or redirected from **stdin** – UEMLoad assumes that input is being supplied from **stdin**, and remains active until it receives an end-of-file indicator.

The UEMLoad utility displays the following message to indicate that it is waiting for input:

`UNV3678I Reading input from stdin. Enter Ctrl+Z <Enter> to cancel wait…`

Cancel the wait by supplying the end-of-file indicator: press <Ctrl+Z> <Enter>.

If a definition load file is provided to UEMLoad via **stdin** redirection, then no prompt is displayed. UEMLoad will read and process the redirected input just as it would a local definition load file.

# 11.2.2  Configuration Options

This section describes the configuration options used to execute UEMLoad for Windows.

## Configuration Options Categories

Table 11.1, below, categorizes the configuration options according to function.

| Category | Description |
|----------|-------------|
| Event Definition | Parameters that correspond to fields in an event definition database record. |
| Event Handler | Parameters that correspond to fields in an event handler database record. |
| General | Affect the overall behavior of the UEMLoad utility. |

Table 11.1  UEMLoad for Windows - Configuration Options Categories

The UEMLoad configuration options for each category are summarized in the following tables.

Each **Option Name** is a link to detailed information about that option in the Universal Event Monitor 3.2.0 Reference Guide.

## Event Definition Options

| Option Name | Description |
|-------------|-------------|
| ACTIVE_DATE_TIME | Date and time at which UEM will begin monitoring an event definition. |
| ASSIGNED_COMPONENT_NAME | Event-driven UEM Server responsible for monitoring the event. |
| EVENT_ID | Identifier that uniquely identifies an event definition record. |
| EVENT_STATE | Event definitions that should be processed or ignored by UEM. |
| EVENT_TYPE | Type of system event represented by the event definition record. |
| EXPIRED_HANDLER_ID | ID of an event handler record that UEM will execute when an event expires. |
| HANDLER_OPTIONS | String that will be passed as command line arguments to the event handler executed by UEM. |
| INACTIVE_DATE_TIME | Date and time at which UEM will stop monitoring an event definition. |
| REJECTED_HANDLER | ID of an event handler record that UEM will execute when an event occurrence is rejected. |
| TRACKING_INTERVAL | Frequency with which UEM will test for the completion of an event occurrence. |
| TRIGGERED_HANDLER_ID | ID of an event handler record that UEM will execute when an event occurrence is triggered. |

### Event Definition Options - Type-Specific
These options are specific to event definitions with an EVENT_TYPE of `FILE`.

| Option Name | Description |
|---|---|
| FILE_SPECIFICATION | Name of a file to monitor. |
| MINIMUM_FILE_SIZE | Size a file must be in order to be considered complete by UEM. |
| RENAME_FILE | Specifies whether or not UEM should rename a monitored file when an event occurrence is triggered. |
| RENAME_FILE_SPECIFICATION | Specifies how a file should be renamed when an event occurrence is triggered. |

# Event Handler Options

| Option Name | Description |
|---|---|
| ENCRYPTION_KEY | Key that was used with Universal Encrypt to encrypt the encrypted user file. |
| HANDLER_ID | Identifier that uniquely identifies an event handler record. |
| HANDLER_TYPE | Type of process executed for the event handler, based on the contents of the USER_COMMAND and USER_SCRIPT parameters |
| MAXIMUM_RETURN_CODE | Highest value with which a handler can exit to still be considered as having executed successfully. |
| USER_COMMAND | Command to execute on behalf of the event handler. |
| USER_FILE_ENCRYPTED | Complete path to a file encrypted with Universal Encrypt. If this file contains a user ID and password, the values for each are stored in the USER_ID and USER_PASSWORD fields, respectively. A UEM Server will re-read this file as it prepares an event handler process for execution, in order to obtain any changes to the user ID and/or password values contained in the file. |
| USER_FILE_PLAIN | Name of a text file that contains user ID and/or password information. |
| USER_ID | ID of a user account in whose security context the handler process will be executed. |
| USER_PASSWORD | Password for the user account specified by the USER_ID parameter. |
| USER_SCRIPT | Text file that contains statements that are executed collectively as a script by UEM. |
| USER_SCRIPT_TYPE | Type of script statements contained in the action field of the event handler record. |
| OPTIONS | String value that is added to the command line UEM Server builds in order to execute an event handler process. The event definition's HANDLER_OPTIONS field is used for the same purpose, but is added to the command string after this field, in order to customize process behavior for that event. |

## General Command Options

| Option Name | Description |
|---|---|
| ACTION | Requested database operation. |
| BROKER_PORT | Port on which a local Universal Broker is accepting incoming connections. |
| COMMAND_ID | Unique command ID associated with the database request. |
| DEFINITION_FILE | Name of a file that contains event definition and/or event handler parameters. |
| HELP | Displays command line help. |
| MESSAGE_LEVEL | Sets the level of messages reported by UEMLoad. |
| VERSION | Displays version information. |

## 11.2.3  Command Line Syntax

Figure 11.1, below, illustrates the command line syntax — using the command line, long form of the configuration options — of UEMLoad for Windows.

```
uemload
-add
{ [-deffile filename] | [eventdefopts [handlerdefopts]] | [handlerdefopts
[eventdefopts]] }
[-cmdid id]
[-level msglevel]
[-port port]

where eventdefopts:
-event_id id
-event_type type
type-specificopts
[-comp_name compname]
[-state state]
[-act_date_time yyyy.mm.dd,hh:mm]
[-inact_date_time yyyy.mm.dd,hh:mm]
[-tracking_int seconds]
[-triggered_id handlerid]
[-rejected_id handlerid]
[-expired_id handlerid]
[-handler_opts options]

where type-specificopts:
{ -event_type FILE -filespec filename [-min_file_size size] [-rename_file
option] [-rename_filespec renamefile] }

where handleropts:
-handler_id id
{ -cmd command | -script filename [-script_type scripttype] }
[ {-file filename | -encryptedfile filename [-key key] | -userid uid
    [-pwd password] } ]
[-handler_type type]
[-maxrc returncode]
[-options cmdlineopts]
```

```
uemload
-delete
[-cmdid id]
[-level msglevel]
[-port port]
{ -deffile filename | -event_id id [-handler_id id ] | -handler_id id
    [-event_id id] }


uemload
-update
{ [-deffile filename] | [eventdefopts [handlerdefopts]] | [handlerdefopts
[eventdefopts]] }
[-cmdid id]
[-level msglevel]
[-port port]


where eventdefopts:
-event_id id
[-event_type type]
[type-specificopts]
[-comp_name compname]
[-state state]
[-act_date_time yyyy.mm.dd,hh:mm]
[-inact_date_time yyyy.mm.dd,hh:mm]
[-tracking_int seconds]
[-triggered_id handlerid]
[-rejected_id handlerid]
[-expired_id handlerid]
[-handler_opts options]


where type-specificopts:
{ -event_type FILE [-filespec filename] [-min_file_size size] [-rename_file
option] [-rename_filespec renamefile] }


where handleropts:
-handler_id id
{ -cmd command | -script filename [-script_type scripttype] }
[ {-file filename | -encryptedfile filename [-key key] | -userid uid
    [-pwd password] } ]
[-handler_type type]
[-maxrc returncode]
[-options cmdlineopts]
```

```
uemload
-list
[-event_id id]
[-comp_name compname]
[-handler_id id]
[-cmdid id]
[-level msglevel]
[-port port]

uemload
-export
[-deffile filename]
[-event_id id]
[-comp_name compname]
[-handler_id id]

uemload
{ -help | -version }
```

Figure 11.1  UEMLoad for Windows - Command Line Syntax

# 11.3  Examples of UEMLoad for Windows

Appendix A Examples provides operating system-specific examples that demonstrate the use of UEM.

Included in this appendix are the following examples that demonstrate the use of UEMLoad for Windows:

- Adding a Single Event Record
- Adding a Single Event Handler Record
- Listing All Event Definitions
- Exporting the Event Definition and Event Handler Databases
- List a Single Event Handler Record
- Listing Multiple Event Definitions and Event Handlers Using Wildcards
- Add Record(s) Using a Definition File
- Add Record(s) Remotely, Using a Definition File Redirected from STDIN
- Add Record(s) Remotely, Using a Definition File Redirected from STDIN (for z/OS)
- Definition File Format

# Chapter 12
# UEMLoad Utility
# for UNIX

## 12.1 Overview

This chapter provides information on Universal Event Monitor (UEM) Load utility, specific to the UNIX operating system.

Note:   Because the UEM Server component is the only UEM component that uses the databases managed by UEMLoad, and because the UEM Server is available only on UNIX and Windows, UEMLoad also is available only on those platforms. The command syntax and general usage of UEMLoad is the same on both platforms, except where noted.

UEMLoad manages records in the event definition and event handler databases. The records in these databases then are used as input to a local UEM Server.

UEMLoad is capable of processing multiple event definition and/or event handler records when the parameters for those records are supplied using a text load file. If no load file is specified, the parameters for a single event definition and/or event handler record can be specified from the command line.

| | |
|---|---|
| **Stoneman's Tip** | Although UEMLoad can access only local event definition and event handler databases, it is possible to store definition load files in a single location (for example, a PDS on z/OS) and distribute them to remote systems via Universal Command. <br><br> Simply redirect the definition load file from stdin and have Universal Command execute UEMLoad on the remote system. UEMLoad will read the redirected input and process it just as it would a local definition load file. This simplifies central administration of remote databases, because definition load files do not have to be stored (and managed) across several systems. |

# 12.2  Usage

The UEMLoad utility executes as a command line application.

The syntax for invoking UEMLoad from the command line requires the name of the program, `uemload`, followed by a list of configuration options.

This section describes the configuration, configuration options, and command line syntax of UEMLoad for Windows.

Section 12.3 Examples of UEMLoad for UNIX provides examples demonstrating the flexibility of UEMLoad.

# 12.2.1 Configuration

Configuration consists of:

- Setting default options and preferences for all executions of UEMLoad.
- Setting options and preferences for a single execution of UEMLoad.

UEMLoad for UNIX receives its configuration options from the following sources:

1. Command line
2. Environment variables

The order of precedence is the same as the list above; command line options being the highest and environment variables being the lowest. That is, options specified via a command line override options specified via environment variables.

See Section 2.2.1 Configuration Methods for complete details on configuration methods and command input for Universal Products.

## Definition and Event Handler Parameters

The configuration options source list, above, applies only to those general command options used to control execution of UEMLoad.

For event definition and/or event handler parameters, input can come from one of three mutually exclusive sources:

1. Command line
2. Definition load file
3. Definition load file redirected from `stdin`

## Definition Load File

When record parameters are provided from the command line, only one event definition and/or event handler can be specified. To add, update, or delete multiple event definition and/or event handler records at the same time, use a definition load file.

If a definition load file is specified from the command line (with the **-deffile** option) along with other event definition and/or event handler record parameters (for example, **-event_id** and **-handler_id**), the UEMLoad request will fail.

In this situation, either:

- Add the event definition and/or event handler parameters to the load file.
- Remove the **-deffile** option from the request.

If no record parameters are specified from a definition load file or from the command line, UEMLoad assumes a definition load file is provided via standard input (that is, **stdin**) redirection.

If UEMLoad detects no input at all — either from the command line or a definition load file stored locally or redirected from **stdin** — UEMLoad assumes that input is being supplied from **stdin**, and remains active until it receives an end-of-file indicator.

The UEMLoad utility displays the following message to indicate that it is waiting for input:

` UNV3678I Reading input from stdin. Enter Ctrl+D to cancel wait…`

Cancel the wait by supplying the end-of-file indicator: press <Ctrl+D>.

If a definition load file is provided to UEMLoad via **stdin** redirection, then no prompt is displayed. UEMLoad will read and process the redirected input just as it would a local definition load file.

## 12.2.2  Configuration Options

This section describes the configuration options used to execute UEMLoad for UNIX.

## Configuration Options Categories

Table 12.1, below, categorizes the configuration options according to function.

| Category | Description |
|----------|-------------|
| Event Definition | Parameters that correspond to fields in an event definition database record. |
| Event Handler | Parameters that correspond to fields in an event handler database record. |
| General | Affect the overall behavior of the UEMLoad utility. |

Table 12.1  UEMLoad for UNIX - Configuration Options Categories

The UEMLoad configuration options for each category are summarized in the following tables.

Each **Option Name** is a link to detailed information about that option in the Universal Event Monitor 3.2.0 Reference Guide.

## Event Definition Options

| Option Name | Description |
|-------------|-------------|
| ACTIVE_DATE_TIME | Date and time at which UEM will begin monitoring an event definition. |
| ASSIGNED_COMPONENT_NAME | Event-driven UEM Server responsible for monitoring the event. |
| EVENT_ID | Identifier that uniquely identifies an event definition record. |
| EVENT_STATE | Event definitions that should be processed or ignored by UEM. |
| EVENT_TYPE | Type of system event represented by the event definition record. |
| EXPIRED_HANDLER_ID | ID of an event handler record that UEM will execute when an event expires. |
| HANDLER_OPTIONS | String that will be passed as command line arguments to the event handler executed by UEM. |
| INACTIVE_DATE_TIME | Date and time at which UEM will stop monitoring an event definition. |
| REJECTED_HANDLER | ID of an event handler record that UEM will execute when an event occurrence is rejected. |
| TRACKING_INTERVAL | Frequency with which UEM will test for the completion of an event occurrence. |
| TRIGGERED_HANDLER_ID | ID of an event handler record that UEM will execute when an event occurrence is triggered. |

### Event Definition Options - Type-Specific

These options are specific to event definitions with an EVENT_TYPE of `FILE`.

| Option Name | Description |
|---|---|
| FILE_SPECIFICATION | Name of a file to monitor. |
| MINIMUM_FILE_SIZE | Size a file must be in order to be considered complete by UEM. |
| RENAME_FILE | Specifies whether or not UEM should rename a monitored file when an event occurrence is triggered. |
| RENAME_FILE_SPECIFICATION | Specifies how a file should be renamed when an event occurrence is triggered. |

# Event Handler Options

| Option Name | Description |
|---|---|
| ENCRYPTION_KEY | Key that was used with Universal Encrypt to encrypt the encrypted user file. |
| HANDLER_ID | Identifier that uniquely identifies an event handler record. |
| HANDLER_TYPE | Type of process executed for the event handler, based on the contents of the USER_COMMAND and USER_SCRIPT parameters |
| MAXIMUM_RETURN_CODE | Highest value with which a handler may exit to still be considered as having executed successfully. |
| USER_COMMAND | Command to execute on behalf of the event handler. |
| USER_FILE_ENCRYPTED | Complete path to a file encrypted with Universal Encrypt. If this file contains a user ID and password, the values for each are stored in the USER_ID and USER_PASSWORD fields, respectively. A UEM Server will re-read this file as it prepares an event handler process for execution, in order to obtain any changes to the user ID and/or password values contained in the file. |
| USER_FILE_PLAIN | Name of a text file that contains user ID and/or password information. |
| USER_ID | ID of a user account in whose security context the handler process will be executed. |
| USER_PASSWORD | Password for the user account specified by the USER_ID parameter. |
| USER_SCRIPT | Text file that contains statements that are executed collectively as a script by UEM. |
| USER_SCRIPT_TYPE | Type of script statements contained in the action field of the event handler record. |
| OPTIONS | String value added to the command line that the UEM Server builds in order to execute an event handler process. The event definition's HANDLER_OPTIONS field is used for the same purpose, but is added to the command string after this field, in order to customize process behavior for that event. |

## General Options

| Option Name | Description |
| --- | --- |
| ACTION | Requested database operation. |
| BROKER_PORT | Port on which a local Universal Broker is accepting incoming connections. |
| COMMAND_ID | Unique command ID associated with the database request. |
| DEFINITION_FILE | Name of a file that contains event definition and/or event handler parameters. |
| HELP | Displays command line help. |
| MESSAGE_LEVEL | Sets the level of messages reported by UEMLoad. |
| VERSION | Displays version information. |

# 12.2.3  Command Line Syntax

Figure 12.1, below, illustrates the syntax of the UEMLoad utility for UNIX.

```
uemload
-add
{ [-deffile filename] | [eventdefopts [handlerdefopts]] | [handlerdefopts
[eventdefopts]] }
[-cmdid id]
[-level msglevel]
[-port port]

where eventdefopts:
-event_id id
-event_type type
type-specificopts
[-comp_name compname]
[-state state]
[-act_date_time yyyy.mm.dd,hh:mm]
[-inact_date_time yyyy.mm.dd,hh:mm]
[-tracking_int seconds]
[-triggered_id handlerid]
[-rejected_id handlerid]
[-expired_id handlerid]
[-handler_opts options]

where type-specificopts:
{ -event_type FILE -filespec filename [-min_file_size size]
  [-rename_file option] [-rename_filespec renamefile] }

where handleropts:
-handler_id id
{ -cmd command | -script filename [-script_type scripttype] }
[ {-file filename | -encryptedfile filename [-key key] | -userid uid
   [-pwd password] } ]
[-handler_type type]
[-maxrc returncode]
[-options cmdlineopts]
```

```
uemload
-delete
[-cmdid id]
[-level msglevel]
[-port port]
{ -deffile filename | -event_id id [-handler_id id ] | -handler_id id
   [-event_id id] }

uemload
-update
{ [-deffile filename] | [eventdefopts [handlerdefopts]] | [handlerdefopts
[eventdefopts]] }
[-cmdid id]
[-level msglevel]
[-port port]

where eventdefopts:
-event_id id
[-event_type type]
[type-specificopts]
[-comp_name compname]
[-state state]
[-act_date_time yyyy.mm.dd,hh:mm]
[-inact_date_time yyyy.mm.dd,hh:mm]
[-tracking_int seconds]
[-triggered_id handlerid]
[-rejected_id handlerid]
[-expired_id handlerid]
[-handler_opts options]

where type-specificopts:
{ -event_type FILE [-filespec filename] [-min_file_size size]
  [-rename_file option] [-rename_filespec renamefile] }

where handleropts:
-handler_id id
{ -cmd command | -script filename [-script_type scripttype] }
[ {-file filename | -encryptedfile filename [-key key] | -userid uid
   [-pwd password] } ]
[-handler_type type]
[-maxrc returncode]
[-options cmdlineopts]
```

```
uemload
-list
[-event_id id]
[-comp_name compname]
[-handler_id id]
[-cmdid id]
[-level msglevel]
[-port port]

uemload
-export
[-deffile filename]
[-event_id id]
[-comp_name compname]
[-handler_id id]

uemload
{ -help | -version }
```

Figure 12.1  UEMLoad Utility for UNIX - Command Line Syntax

# 12.3  Examples of UEMLoad for UNIX

Appendix A Examples provides operating system-specific examples that demonstrate the use of UEM.

Included in this appendix are the following examples that demonstrate the use of UEMLoad for UNIX:

- Adding a Single Event Record
- Adding a Single Event Handler Record
- Listing All Event Definitions
- Exporting the Event Definition and Event Handler Databases
- List a Single Event Handler Record
- Listing Multiple Event Definitions and Event Handlers Using Wildcards
- Add Record(s) Using a Definition File
- Add Record(s) Remotely, Using a Definition File Redirected from STDIN
- Add Record(s) Remotely, Using a Definition File Redirected from STDIN (for z/OS)
- Definition File Format

# Appendix A
# Examples

## A.1  Overview

This appendix provides operating system-specific examples that demonstrate the use of UEM:

- UEM Manager for z/OS Examples
- UEM Manager for Windows Examples
- UEM Manager for UNIX Examples
- UEMLoad for Windows Examples
- UEMLoad for UNIX Examples

# A.2  UEM Manager for z/OS Examples

This section provides examples that demonstrate the use of UEM Manager for z/OS.

They assume the following information:

- UEM Server is installed on a remote system named `uemhost`.
- Security option has been enabled in the UEM Server's configuration.

The values for the `-userid` and `-pwd` parameters represent the user ID and password of a valid user account defined on `uemhost`.

The following list provides a link to each example.

- Starting a UEM (Event-Driven) Server
- Refreshing a UEM Server (Event-Driven)
- Using a Stored Event Handler Record
- Handling an Event With a Script
- Handling an Expired Event
- Continuation Character "-" in z/OS Handler Script
- Continuation Character "+" in z/OS Handler Script
- Continuation Characters "-" and "+" in z/OS Handler Script

# A.2.1  Starting a UEM (Event-Driven) Server

There are two ways start a UEM event-driven Server (`uems`) component:

1. Recycle the `ubroker` daemon (Unversal Broker service under Windows).
2. Use Universal Control to start the `uems`, either locally on the server or from the mainframe.

In this example, `uems` is started from the mainframe.

(This job will fail if `uems` is running at the time of submit; `uems` usually is started by the Universal Broker when it is started.)

```
//STUEMS    JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//          JCLLIB ORDER=SBI.UNV.SUNVSAMP
//*
//STEP1     EXEC UCTLPRC
//LOGONDD   DD  DISP=SHR,DSN=MFC1A.JCL.CNTRL(WINUSER)
//SYSIN     DD  *
-host 172.16.30.30 -x LOGONDD -port 7887 -start uems
/*
```

Figure A.1  Starting a UEM Event-Driven Server

Note:  There is only one different command (`-start`) between this example and Refreshing a UEM Server (Event-Driven).

# A.2.2  Refreshing a UEM Server (Event-Driven)

In this example, RESUEMS will refresh the UEM event-drtiven Server (`uems`) to secure changes made to the configuration file.

```
//RESUEMS   JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//          JCLLIB ORDER=SBI.UNV.SUNVSAMP
//*
//STEP1     EXEC UCTLPRC
//LOGONDD  DD  DISP=SHR,DSN=MFC1A.JCL.CNTRL(WINUSER)
//SYSIN    DD  *
-host 172.16.30.30 -x LOGONDD -port 7887 -refresh uems
/*
```

Figure A.2  Refreshing a UEM Event-Driven Server

Note:   There is only one different command (`-refresh`) between this example and
        Starting a UEM (Event-Driven) Server.

# A.2.3  Using a Stored Event Handler Record

In this example, a demand-driven UEM Server will watch for the creation of a file called **uemtest.dat**. Since no path is specified, it will look for this file in the user's UEM Server working directory, as specified in the component definition for a demand-driven UEM Server.

If the file completes before the inactive time of *17:38* elapses, the event occurrence will be set to the **triggered** state, and UEM will execute the command or script contained in the event handler **h001**, which is the ID of a record in the event handler database.

If the file does not complete before the inactive time elapses, the event occurrence will be set to a **rejected** state. Since no event handler information is provided for a rejected occurrence, no further action will be taken by the UEM Server.

If the UEM Server does not detect the presence of **uemtest.dat** before the inactive time of *17:38* elapses, the event will be set to an **expired** state.

Note:   Because the inactive date value was omitted, UEM Manager will default the inactive date to the current date. Further, because no handler information is given for the **expired** state, no further action will be taken by the UEM Server once the event expires.

```
//jobname   JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//STEP1     EXEC UEMPRC
//SYSIN     DD *
 -event_type        file
 -filespec          uemtest.dat
 -wait              yes
 -inact_date_time   ,17:38
 -triggered_id      h001
 -host              uemhost
 -userid            uemuser
 -pwd               uemusers_password
 -max_count         1
/*
```

Figure A.3  Using a Stored Event Record

# A.2.4  Handling an Event With a Script

In this example, a demand-driven UEM Server installed on a Windows machine will watch for the creation of a file called `uemtest.dat`. Since no path is specified, it will look for this file in the user's UEM Server working directory.

A relative inactive date/time is used to instruct the UEM Server to monitor the event for *10* minutes. If the file is detected and completes within that time, the event occurrence will be set to the `triggered` state. The script statements contained within the `MYSCRIPT` DD statement then will be written to a temporary script file and executed by UEM Server.

The value specified by the `-handler_opts` option is appended to the command line constructed by UEM in order to execute the temporary script file. This will cause the values *parm1*, *parm2*, and *parm3* to be passed to the script. Further, any output generated by the script will be written to a file in the UEM Server working directory, `uemtest.log`.

If the file is detected, but does not complete before the inactive time elapses, the event occurrence will be set to a `rejected` state. Since no event handler information is provided for a rejected occurrence, no further action will be taken by the UEM Server.

If the UEM Server does not detect the presence of `uemtest.dat` before the inactive time elapses, the event will be set to an `expired` state. Again, because no handler information is given for this state, no further action will be taken by the UEM Server.

```
//jobname   JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//STEP1     EXEC UEMPRC
//MYSCRIPT DD *
@echo off

:: Program variables
set parmCtr=1

:: Loop through parameter list

:: **** Start of loop ****
:BeginLoop
if ""%1""=="""" goto EndLoop

:DisplayParm
echo Parm %parmCtr%: %1

:: Shift the next parm
shift
set /a parmCtr+=1

:: Go back to the top
goto BeginLoop

:: **** End of loop ****
:EndLoop

//SYSIN    DD *
 -event_type       file
 -filespec         uemtest.dat
 -inact_date_time  +10
 -handler_opts     "parm1 parm2 parm3 >uemtest.log 2>&1"
 -host             uemhost
 -userid           uemuser
 -pwd              uemusers_password
 -triggered -script myscript
/*
```

Figure A.4  Handling an Event with a Script

# A.2.5 Handling an Expired Event

In this example, a demand-driven UEM Server installed on a UNIX system watches for the creation of a file called **uemtest.dat**. The **-filespec** option contains no path information, so UEM Server looks for this file in uemuser's home directory.

A relative inactive date / time instructs the demand-driven Server to monitor the event for one (1) minute. If the UEM Server detects the file, and the file completes within that time, UEM sets the event occurrence to the triggered state. Since the command options contain no event handler information for a triggered occurrence, the UEM Server simply renames the file (by default). UEM Server then makes the event inactive, and ends.

If the UEM Server detects the file, but the file does not complete within 60 seconds, UEM sets the occurrence to the rejected state. Since the command options contain no event handler information for a rejected occurrence, the UEM Server leaves the file as-is and takes no further action.

If the UEM Server does not detect the presence of uemtest.dat before the inactive time elapses, it sets the event to the expired state. When this happens, the UEM Server executes the command specified by the **-cmd** parameter of the **-expired** option. In this example, UEM executes the **ls -alR /home** command.

Note that the **-expired** option also contains the **-options** parameter. In this example, the **-options** parameter redirects the output of the **ls -alR /home** command to a file in uemuser's home directory, **uemtest.log**.

```
//jobname   JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//STEP1     EXEC UEMPRC
//SYSIN     DD *
 -event_type      file
 -filespec        uemtest.dat
 -inact_date_time +1
 -expired -cmd "ls -alR /home" -options ">uemtest.log 2>&1"
 -host            uemhost
 -userid          uemuser
 -pwd             uemusers_password
/*
```

Figure A.5  Handling an Expired Event

# A.2.6 Continuation Character "-" in z/OS Handler Script

Continuation characters ("-" and "+") are useful when you want to execute a script line that is longer than your available z/OS character space.

The "-" continuation character will preserve trailing spaces in your line.

The "+" continuation character will not preserve trailing spaces in your line.

---

**The following z/OS handler script:**

```
begin_script
 stmt "ls -a -            <---- Notice the continuation character "-"
 >dirfile"
 end_script
```

**Will produce the following output when loaded to the uemh.db:**

```
Handler ID...................: MFCTRIGGER_1
Handler Type.................: SCRIPT
Max Acceptable Return Code...: 0
Encrypted User File..........:
User ID......................: mfc1a
Script statements:
 ls -a  >dirfile
 Script Type.................: bat
Command Line Options.........:
Last Modified On.............: 06/11/09 10:32:31 AM
Last Modified By.............: mfc1a
```

---

Figure A.6  Continuation Character "-"  in z/OS Handler Script

# A.2.7  Continuation Character "+" in z/OS Handler Script

Continuation characters  ("-" and "+") are useful when you want to execute a script line that is longer than your available z/OS character space.

The "-" continuation character will preserve trailing spaces in your line.

The "+" continuation character will not preserve trailing spaces in your line.

---

**The following z/OS handler script:**

```
begin_script
 stmt "ls -a >dir +             <---- Notice the continuation character "+"
 file"
 end_script
```

**Will produce the following output when loaded to the uemh.db:**

```
Handler ID...................: MFCTRIGGER_1
Handler Type.................: SCRIPT
Max Acceptable Return Code...: 0
Encrypted User File..........:
User ID......................: mfc1a
Script statements:
 ls -a >dirfile
Script Type..................: bat
Command Line Options.........:
Last Modified On.............: 06/11/09 11:46:32 AM
Last Modified By.............: mfc1a
```

---

Figure A.7  Continuation Character "+" in z/OS Handler Script

# A.2.8  Continuation Characters "-" and "+" in z/OS Handler Script

Continuation characters  ("-" and "+") are useful when you want to execute a script line that is longer than your available z/OS character space.

The "-" continuation character will preserve trailing spaces in your line.

The "+" continuation character will not preserve trailing spaces in your line.

This example shows the use of "+" to concatenate a command line or a word within a z/OS script without a space as the use of "-" to continue a line of script where a space is required within the same z/OS handler script.

---

**The following z/OS handler script:**
```
 begin_script
 stmt "ls -a >dir +
 file"
 stmt "uemFName=`basename \u201c$UEMORIGFILE\u201d | sed 's/\(.*\)+
  \.\(.*$\)/\1/'`"
 stmt "fname=$uemFName.$dt.$tm.$pid.txt"
 stmt " ls -al >dir+
 data"
 stmt "ls -a -
 >new+
data"
 end_script
```


 **Will produce the following output when loaded to the uemh.db:**
```
Handler ID..................: MFCTRIGGER_1
Handler Type................: SCRIPT
Max Acceptable Return Code...: 0
Encrypted User File.........:
User ID.....................: mfc1a
Script statements:
 ls -a >dirfile
 uemFName=`basename \u201c$UEMORIGFILE\u201d | sed 's/\(.*\)\.\(.*$\)/\1/'`
 fname=$uemFName.$dt.$tm.$pid.txt
  ls -al >dirdata
 ls -a  >newdata
Script Type.................: bat
Command Line Options........:
Last Modified On............: 06/11/09 01:25:20 PM
Last Modified By............: mfc1a
```

Figure A.8  Continuation Characters "-" and "+" in z/OS Handler Script

---

# A.3  UEM Manager for Windows Examples

This section contains examples demonstrating the use of UEM Manager for Windows.

The examples assume that:

- UEM Server is installed on a remote system named `uemhost`.
- Security option has been enabled in the UEM Server's configuration.

The values for the `-userid` and `-pwd` parameters represent the user ID and password of a valid user account defined on `uemhost`.

The following list provides a link to each example.

- Using a Stored Event Handler Record
- Executing a Script for a Triggered Event Occurrence
- Handling an Expired Event

## A.3.1  Using a Stored Event Handler Record

In this example, a demand-driven UEM Server will watch for the creation of a file called **uemtest.dat** in the **C:\UEM Files** directory.

If the file completes before the inactive time of *20:00* elapses, the event occurrence will be set to the **triggered** state, and UEM will execute the command or script contained in the event handler **h001**, which is the ID of a record in the event handler database.

If the file does not complete before the inactive time elapses, the event occurrence will be set to a rejected state. Since no event handler information is provided for a rejected occurrence, no further action will be taken by the UEM Server.

If the UEM Server does not detect the presence of **C:\UEM Files\uemtest.dat** before the inactive time of *20:00* elapses, the event will be set to an **expired** state.

Note:   Because the inactive date value was omitted, UEM Manager will default the inactive date to the current date. Further, because no handler information is given for the **expired** state, no further action will be taken by the UEM Server once the event expires.

```
uem -host uemhost -event_type file
-filespec "C:\UEM Files\uemtest.dat"
-inact_date_time ,20:00 -userid uemuser -pwd uemusers_password
-triggered -handler_id h001
```
Figure A.9  Using a Stored Event Handler Record

## A.3.2  Executing a Script for a Triggered Event Occurrence

In this example, a demand-driven UEM Server installed on a UNIX machine will watch for the creation of a file called `uemtest.dat`. Since no path is specified, it will look for this file in the user's home directory.

A relative inactive date/time is used to instruct the UEM Server to monitor the event for *10* minutes. If the file is detected and completes within that time, the event occurrence will be set to the `triggered` state. The script statements contained within the local file `C:\UEMScripts\h_001.txt` then will be written to a temporary script file on `uemhost` and executed by UEM Server. The value specified by the `-handler_opts` option is appended to the command line constructed by UEM to execute the temporary script file. This will cause the values *parm1*, *parm2*, and *parm3* to be passed to the script. Any output generated by the script will be written to a file in the UEM Server working directory called `uemtest.log`.

If the file is detected, but does not complete before the inactive time elapses, the event occurrence will be set to a `rejected` state. Since no event handler information is provided for a rejected occurrence, no further action will be taken by the UEM Server.

If the UEM Server does not detect the presence of `uemtest.dat` before the inactive time elapses, the event will be set to an `expired` state. Again, because no handler information is given for this state, no further action will be taken by the UEM Server.

```
uem -host uemhost -event_type file -filespec uemtest.dat
-inact_date_time +10 -userid uemuser -pwd uemusers_password
-triggered -script C:\UEMScripts\h_001.txt
-handler_opts "parm1 parm2 parm3 >uemtest.log 2>&1"
```

Figure A.10  Handling an Event with a Script.

Figure A.11, below, illustrates the contents of the `C:\UEMScripts\h_001.txt` file.

```
#!/bin/sh

# Sample script h_001.txt

argNum=1

# Display each command line argument.
while [ "$1" != "" ]
do
echo Parm $argNum: $1
shift
argNum=`expr $argNum + 1`
done
```

Figure A.11  Contents of Sample Script File

# A.3.3  Handling an Expired Event

In this example, a demand-driven UEM Server installed on a UNIX system watches for the creation of a file called **uemtest.dat** in the **/uem** files directory.

Note the space that precedes the path name specified in the **-filespec** parameter. This is necessary to accommodate parsing requirements for command options in Windows (see the FILE_SPECIFICATION option in the Universal Event Monitor 3.2.0 Reference Guide).

A relative inactive date / time instructs the demand-driven Server to monitor the event for one (1) minute. If the UEM Server detects the file, and the file completes within that time, UEM sets the event occurrence to the triggered state. Since the command options contain no event handler information for a triggered occurrence, the UEM Server simply renames the file (by default). UEM Server then makes the event inactive, and ends.

If the UEM Server detects the file, but the file does not complete within 60 seconds, UEM sets the occurrence to the rejected state. Since the command options contain no event handler information for a rejected occurrence, the UEM Server leaves the file as-is and takes no further action.

If the UEM Server does not detect the presence of uemtest.dat before the inactive time elapses, it sets the event to the expired state. When this happens, the UEM Server executes the command specified by the **-cmd** parameter of the **-expired** option. In this example, UEM executes the **'ls -alR /uem files'** command.

Note that the **-expired** option also contains the **-options** parameter. In this example, the **-options** parameter redirects the output of the **'ls -alR /uem files'** command to a file in uemuser's home directory called **uemtest.log**.

```
uem -host uemhost -event_type file
-userid uemuser -pwd uemusers_password
-filespec " /uem files/uemtest.dat"
-inact_date_time +1
-expired -cmd "ls -alR '/uem files'" -options ">uemtest.log 2>&1"
```

Figure A.12  Handling an Expired Event

# A.4 UEM Manager for UNIX Examples

This section contains examples demonstrating the use of UEM Manager for UNIX.

The examples assume that:

- UEM Server is installed on a remote system named `uemhost`.
- Security option has been enabled in the UEM Server's configuration.

The values for the `-userid` and `-pwd` parameters represent the user ID and password of a valid user account defined on `uemhost`.

The following list provides a link to each example.

- Using a Stored Event Handler Record
- Executing a Script for a Triggered Event Occurrence
- Handling an Expired Event

# A.4.1  Using a Stored Event Handler Record

In this example, a UEM Server (installed on a Windows system) will watch for the creation of a file called `uemtest.dat` in the `C:\UEM Files` directory.

If the file completes before the inactive time of *08:00* elapses, the event occurrence will be set to the `triggered` state. UEM then will execute the command or script contained in the event handler `h001`, which is the ID of a record in the event handler database.

If the file does not complete before the inactive time elapses, the event occurrence will be set to a `rejected` state. Since no event handler information is provided for a rejected occurrence, no further action will be taken by the UEM Server.

If the UEM Server does not detect the presence of `C:\UEM Files\uemtest.dat` before the inactive time of *08:00* elapses, the event will be set to an `expired` state.

Note:    Because the inactive date value was omitted, UEM Manager will default the inactive date to the current date. Again, because no handler information is given for the `expired` state, no further action will be taken by the UEM Server once the event expires.

```
uem -host uemhost -event_type file
-filespec "C:\UEM Files\uemtest.dat"
-inact_date_time ,08:00 -userid uemuser -pwd uemusers_password
-triggered -handler_id h001
```

Figure A.13  Using a Stored Event Handler Record

## A.4.2  Executing a Script for a Triggered Event Occurrence

In this example, a UEM Server installed on a Windows machine will watch for the creation of a file called **uemtest.dat**. Since no path is specified, it will look for this file in the user's UEM Server working directory.

A relative inactive date/time is used to instruct the UEM Server to monitor the event for *10* minutes. If the file is detected and completes within that time, the event occurrence will be set to the **triggered** state. The script statements contained within the local file **/UEMScripts/h_001.txt** then will be written to a temporary script file on **uemhost** and executed by the UEM Server. The value specified by the **-handler_opts** option is appended to the command line constructed by UEM to execute the temporary script file. This will cause the values *parm1*, *parm2*, and *parm3* to be passed to the script. Any output generated by the script will be written to a file in the UEM Server working directory called **uemtest.log**.

If the file is detected, but does not complete before the inactive time elapses, the event occurrence will be set to a **rejected** state. Since no event handler information is provided for a rejected occurrence, no further action will be taken by the UEM Server.

If the UEM Server does not detect the presence of **uemtest.dat** before the inactive time elapses, the event will be set to an **expired** state. Again, because no handler information is given for this state, no further action will be taken by the UEM Server.

```
uem -host uemhost -event_type file -filespec uemtest.dat
-inact_date_time +10 -userid uemuser -pwd uemusers_password
-triggered -script /UEMScripts/h_001.txt
-handler_opts "parm1 parm2 parm3 >uemtest.log 2>&1"
```

Figure A.14  Handling an Event with a Script

Figure A.15, below, illustrates the contents of the **/UEMScripts/h_001.txt** file.

```
:: Sample script h_001.txt
@echo off

:: Program variables
set parmCtr=1

:: Loop through parameter list

:: **** Start of loop ****
:BeginLoop
if ""%1""=="""" goto EndLoop

:DisplayParm
echo Parm %parmCtr%: %1

:: Shift the next parm
shift
set /a parmCtr+=1

:: Go back to the top
goto BeginLoop

:: **** End of loop ****
:EndLoop
```

Figure A.15  Contents of Sample Script File

# A.4.3 Handling an Expired Event

In this example, a demand-driven UEM Server (installed on a different UNIX system) watches for the creation of a file called `uemtest.dat`. The `-filespec` option contains no path information, so UEM Server looks for this file in uemuser's home directory.

A relative inactive date/time instructs the demand-driven Server to monitor the event for one (1) minute. If the UEM Server detects the file, and the file completes within that time, UEM sets the event occurrence to the triggered state. Since the command options contain no event handler information for a triggered occurrence, the UEM Server simply renames the file (by default). UEM Server then makes the event inactive, and ends.

If the UEM Server detects the file, but the file does not complete within 60 seconds, UEM sets the occurrence to the rejected state. Since the command options contain no event handler information for a rejected occurrence, the UEM Server leaves the file as-is and takes no further action.

If the UEM Server does not detect the presence of uemtest.dat before the inactive time elapses, it sets the event to the expired state. When this happens, the UEM Server executes the command specified by the `-cmd` parameter of the `-expired` option. In this example, UEM executes the `'ls -alR /uem files'` command.

Note that the `-expired` option also contains the -options parameter. In this example, the `-options` parameter redirects the output of the `'ls -alR /uem files'` command to a file in uemuser's home directory called `uemtest.log`.

```
uem –host uemhost –event_type file –filespec uemtest.dat
-userid uemuser –pwd uemusers_password
-inact_date_time +1
-expired -cmd 'ls –alR "/uem files"' -options '>uemtest.log 2>&1'
```

Figure A.16  Handling an Expired Event

# A.5 UEMLoad for Windows Examples

This section contains examples demonstrating the use of UEMLoad for Windows.

It also provides a sample definition file.

The following list provides a link to each example.

- Adding a Single Event Record
- Adding a Single Event Handler Record
- Listing All Event Definitions
- Exporting the Event Definition and Event Handler Databases
- List a Single Event Handler Record
- Listing Multiple Event Definitions and Event Handlers Using Wildcards
- Add Record(s) Using a Definition File
- Add Record(s) Remotely, Using a Definition File Redirected from STDIN
- Definition File Format

# A.5.1 Adding a Single Event Record

In this example, a single event record identified as `payrollfile` is added to the local event definition database.

This event definition will instruct a UEM Server, which resides on the local (UNIX) system, to detect all occurrences of the file `/tmp/payroll.dly`. Whenever UEM detects this file and sets the associated event occurrence to a `triggered` state, UEM will execute the command or script contained in the stored event handler record that has an ID of `listdir`. If this event handler record does not exist at the time the event occurrence is triggered, an error will be issued by UEM.

When the record has been added to the event definition database, it is immediately available for use by a demand-driven UEM Server. In other words, there is no restriction with respect to how quickly a UEM Manager can reference the stored event definition after UEMLoad adds it to the database.

Because no values for the EVENT_STATE, ACTIVE_DATE_TIME, and INACTIVE_DATE_TIME options were specified, the default values of *ENABLED*, the current date and time, and *2038.01.16,23:59*, respectively are used. This means the event will be monitored as soon as the event definition is assigned to an event-driven UEM Server. In this case, the event definition is assigned to the UEM Server component with an ID of `uems` (the default).

If this UEM Server component is active when the record is added, this assignment will occur the next time that the UEM Server refreshes its configuration. If the UEM Server component is not active, the assignment is made the next time it is started.

```
uemload -add -event_id payrollfile -event_type file
-filespec "/tmp/payroll.dly" -triggered_id listdir
```

Figure A.17  Adding a single event definition record.

## A.5.2  Adding a Single Event Handler Record

In this example, a single handler record identified, `listdir`, is added to the local event handler database.

Whenever a UEM Server invokes this event handler, a handler process is started that executes the command `ls -al`, which lists the contents of the current directory on a UNIX system. The `encrypted.file` file, referenced by the `-encryptedfile` parameter, is a Universal Encrypted file. This file contains a user ID and, optionally, a password that is used by an event-driven UEM Server to establish a security context in which to execute the handler process (provided the USER_SECURITY option is enabled in the UEM Server's configuration).

Once this record is added, it is available immediately to both demand-driven and event-driven UEM Servers.

Note:  If a demand-driven UEM Server uses this handler, any user information specified in `encrypted.file` is overridden by the user information provided by the UEM Manager's command options.

```
uemload –add –handler_id listdir –encryptedfile encrypted.file
–cmd "ls –al"
```

Figure A.18  Adding a single event handler record.

## A.5.3  Listing All Event Definitions

In this Windows example, the **-list** option is used to dump all records in the event definition database and display them to **stdout**.

If the request were executed on a UNIX system, the asterisk ( **\*** ) would need to be escaped or enclosed within quotes (that is: **\\\*** or "**\***", respectively).

```
uemload -list -event_id *
```
Figure A.19  Listing all event definition records.


Note:   The default behavior when **list**ing or **export**ing records, when neither an event ID nor a handler ID is specified, is to return all records. However, in this example above, even though no handler ID was specified, no event handler records are returned.

Conversely, if just a handler ID had been specified, no event definition records would be returned. Supplying an event ID and/or handler ID serves as a filter which causes **uemload** to return just those records specifically requested.

# A.5.4  Exporting the Event Definition and Event Handler Databases

In this example, the `-export` option is used to dump all records in the event definition and event handler databases to a text file in the current directory named `uemout.txt`. This file is a UEMLoad definition file that also can be used to add or update records in the event definition and/or event handler databases.

The contents of the file resembles the examples shown in Figure A.27.

```
uemload -export -deffile uemout.txt
```

Figure A.20  Exporting all Event and Handler Records

Note:   No event ID or handler ID is specified from the command line. If neither parameter is specified when `list`ing or `export`ing records, the default behavior is to retrieve all database records.

# A.5.5 List a Single Event Handler Record

In this example, the **-list** option is used to display the contents of an event handler record with an ID of **dirlist**.

```
uemload -list -handler_id dirlist
```

Figure A.21  List a Single Event Handler Record

Figure A.22, below, illustrates sample output for this command. (The values shown are those that could be expected if the record were added using the command shown in Figure A.17.)

In this specific instance, the user ID contained in **encrypted.file** (from Figure A.17) is **sparkie**, and the record was added by the user account with an ID of **sbuser**.

```
UNV3659I Connecting to local broker on port 7887.
UNV3406I Universal Event Monitor Server component 1117035117 started.
UNV3666I Load request started at 11:32:45 AM 05/25/2005.

Event Handler(s):
=================
Handler ID................: dirlist
Max Acceptable Return Code.: 0
User ID....................: sparkie
Command....................: ls -al
Last Modified On...........: 05/25/2005 11:32:06 AM
Last Modified By...........: sbuser

UNV3667I Universal Event Monitor Load is ending successfully with exit code 0.
```

Figure A.22  Sample List Output

# A.5.6  Listing Multiple Event Definitions and Event Handlers Using Wildcards

In this example, the wildcards supported by `uemload` are demonstrated.

Wildcards can be used to select event definitions and event handlers whose respective IDs match the specified pattern.

- Asterisk ( **\*** ) can be used to match 0 or more characters.
- Question mark ( **?** ) can be used to match any single character.

All event definitions whose IDs start with the characters `event` are returned by the command below. In addition, all event handlers whose IDs begin with `handler0` and end with any two characters are selected.

```
uemload -list -event_id event* -handler_id handler0??
```

Figure A.23  Using Wildcards to List Records

# A.5.7  Add Record(s) Using a Definition File

In this example, a text file named `uemadd.txt` is used to add one or more records to the UEM databases. The contents of the file resemble those shown in Figure A.27.

A definition file allows multiple records to be added to the event definition and/or event handler databases at the same time. When no definition file is used, only a single record can be added to the database(s).

```
uemload -add -deffile uemadd.txt
```

Figure A.24  Add Database Record(s) Using a Definition File

# A.5.8  Add Record(s) Remotely, Using a Definition File Redirected from STDIN

In this example, a definition load file named **`uemadd.txt`** is used to add one or more records to the databases of a remote UEM Server. The contents of the file resemble those shown in Figure A.27.

Universal Command is used to execute UEMLoad on the remote UEM Server's system. The definition load file is redirected from standard input (**`stdin`**), which eliminates the step of copying the load file to the remote system before executing UEMLoad.

```
ucmd –cmd "uemload –add" –host rmthost –x rmtacctinfo.enc <uemadd.txt
```

Figure A.25  Redirect Definition File from stdin

## A.5.9  Add Record(s) Remotely, Using a Definition File Redirected from STDIN (for z/OS)

In this example, a definition load file named **MY.UEM.DATA(UEMDEF)** is used to add one or more records to the databases of a remote UEM Server. The contents of the file resemble those shown in Figure A.27.

Universal Command is used to execute UEMLoad on the remote UEM Server's system. It redirects standard input (stdin) from a data set allocated to the UNVIN ddname. This eliminates the step of copying the data set to the remote system before executing UEMLoad.

```
//STEP1     EXEC UCMDPRC
//UNVIN     DD  DISP=SHR,DSN=MY.UEM.DATA(UEMDEF)
//SYSIN     DD  *
 -HOST              dallas
 -USERID            joe
 -PWD               ahzidaeh
 -CMD               "uemload -add"
```

Figure A.26  Redirect Definition File from STDIN (for z/OS)

# A.5.10 Definition File Format

The format of the definition load file for events and event handlers follows the keyword / value-pair convention used for Universal Products configuration files. However, because more than one definition can be specified in a load file, some additional conventions are used.

The **begin_event** and **end_event** keywords are used to mark the beginning and end, respectively, of each event definition entry.

The **begin_handler** and **end_handler** keywords are used to mark the beginning and end, respectively, of each event handler entry.

The **begin_script** and **end_script** keywords are used to mark the beginning and end, respectively, of any user script contained in the definition load file.

Lines that belong to the script must begin with the **stmt** keyword. Long **stmt** values that have to be split across lines can be done so using the **+** and **−** line continuation characters (as described in Section 2.2.6 Configuration File Syntax). These lines will be accepted verbatim, and no script syntax validation will be done. Lines will continue to be added to the script until an **end_script**, **end_handler**, **begin_handler**, or **begin_event** keyword is read, or the end of the file is reached.

If a parameter's value contains spaces, it must be enclosed in double ( **"** ) quotation marks.

If quotes are to be saved as part of the parameter's value, use extra double ( **"** ) quotation marks to escape the quotes (for example, **optname** "**optval1** ""**optval2 optval2a**"" **optval3**").

The **script** keyword can be used in lieu of a **begin_script**/**end_script** block, in which case the contents of the specified file will be written to the event handler.

A sample definition file for Windows is shown in Figure A.27.

```
# Indented lines are for illustration only. Leading spaces are
# ignored by UEMLoad. Defaults will be used for any omitted
# values.

# Start of parameters for an event definition with an ID of
# "win_event_sample".

begin_event
   event_id win_event_sample
   event_type FILE
   comp_name uems
```

```
      state enable
      inact_date_time 2004.12.31,23:59
      triggered_id script_sample
      filespec "uem*.dat"
      rename_file yes
      rename_filespec "$(compname).$(compid).$(date).$(seqno)"
end_event


# End of parameters for event definition "win_event_sample".


# Start of parameters for an event handler with an ID of
# "win_script_sample".


begin_handler
      handler_id script_sample
      handler_type SCRIPT
      maxrc 0
      userid uemuser
      begin_script
          stmt "@echo off"
          stmt ""
          stmt "dir /-p/o/s ""C:\Program Files"""
      end_script
      script_type bat
end_handler


# End of parameters for event handler "win_script_sample".



# Start of parameters for an event definition with an ID of
# "win_cmd_sample".


begin_handler
      handler_id cmd_sample
      maxrc 0
      userid uemuser
      cmd "C:\Documents and Settings\uemuser\TEST.BAT"
end_handler


# End of parameters for event definition "win_cmd_sample".
```

Figure A.27  Definition File Sample - Windows

# A.6  UEMLoad for UNIX Examples

This section contains examples demonstrating the use UEMLoad for Windows.

It also provides a sample definition file.

The following list provides a link to each example.

- Adding a Single Event Record
- Adding a Single Event Handler Record
- Listing All Event Definitions
- Exporting the Event Definition and Event Handler Databases
- List a Single Event Handler Record
- Listing Multiple Event Definitions and Event Handlers Using Wildcards
- Add Record(s) Using a Definition File
- Add Record(s) Remotely, Using a Definition File Redirected from STDIN
- Add Record(s) Remotely, Using a Definition File Redirected from STDIN (for z/OS)
- Definition File Format

# A.6.1  Adding a Single Event Record

In this example, a single event record identified as `payrollfile` is added to the local event definition database.

This event definition will instruct a UEM Server, which resides on the local (UNIX) system, to detect all occurrences of the file `/tmp/payroll.dly`. Whenever UEM detects this file and sets the associated event occurrence to a `triggered` state, UEM will execute the command or script contained in the stored event handler record that has an ID of `listdir`. If this event handler record does not exist at the time the event occurrence is triggered, an error will be issued by UEM.

When the record has been added to the event definition database, it is immediately available for use by a demand-driven UEM Server. In other words, there is no restriction with respect to how quickly a UEM Manager can reference the stored event definition after UEMLoad adds it to the database.

Because no values for the EVENT_STATE, ACTIVE_DATE_TIME, and INACTIVE_DATE_TIME options were specified, the default values of *ENABLED*, the current date and time, and *2038.01.16,23:59*, respectively are used. This means the event will be monitored as soon as the event definition is assigned to an event-driven UEM Server. In this case, the event definition is assigned to the UEM Server component with an ID of `uems` (the default).

If this UEM Server component is active when the record is added, this assignment will occur the next time that the UEM Server refreshes its configuration. If the UEM Server component is not active, the assignment is made the next time it is started.

```
uemload -add -event_id payrollfile -event_type file
-filespec "/tmp/payroll.dly" -triggered_id listdir
```

Figure A.28  Adding a single event definition record.

# A.6.2  Adding a Single Event Handler Record

In this example, a single handler record identified, `listdir`, is added to the local event handler database.

Whenever a UEM Server invokes this event handler, a handler process is started that executes the command `ls -al`, which lists the contents of the current directory on a UNIX system. The `encrypted.file` file, referenced by the `-encryptedfile` parameter, is a Universal Encrypted file. This file contains a user ID and, optionally, a password that is used by an event-driven UEM Server to establish a security context in which to execute the handler process (provided the USER_SECURITY option is enabled in the UEM Server's configuration).

Once this record is added, it is available immediately to both demand-driven and event-driven UEM Servers.

Note:   If a demand-driven UEM Server uses this handler, any user information specified in `encrypted.file` is overridden by the user information provided by the UEM Manager's command options.

```
uemload -add -handler_id listdir -encryptedfile encrypted.file
-cmd "ls -al"
```

Figure A.29  Adding a single event handler record.

## A.6.3  Listing All Event Definitions

In this Windows example, the **-list** option is used to dump all records in the event definition database and display them to **stdout**.

If the request were executed on a UNIX system, the asterisk ( * ) would need to be escaped or enclosed within quotes (that is: \* or "*", respectively).

```
uemload -list -event_id *
```

Figure A.30  Listing all event definition records.

Note:  The default behavior when **list**ing or **export**ing records, when neither an event ID nor a handler ID is specified, is to return all records. However, in this example above, even though no handler ID was specified, no event handler records are returned.

Conversely, if just a handler ID had been specified, no event definition records would be returned. Supplying an event ID and/or handler ID serves as a filter which causes **uemload** to return just those records specifically requested.

# A.6.4  Exporting the Event Definition and Event Handler Databases

In this example, the `-export` option is used to dump all records in the event definition and event handler databases to a text file in the current directory named `uemout.txt`. This file is a UEMLoad definition file that also can be used to add or update records in the event definition and/or event handler databases.

The contents of the file resembles the examples shown in Figure A.27.

```
uemload –export –deffile uemout.txt
```
Figure A.31  Exporting all Event and Handler Records

Note:   No event ID or handler ID is specified from the command line. If neither parameter is specified when `list`ing or `export`ing records, the default behavior is to retrieve all database records.

# A.6.5  List a Single Event Handler Record

In this example, the **-list** option is used to display the contents of an event handler record with an ID of **dirlist**.

```
uemload -list -handler_id dirlist
```

Figure A.32  List a Single Event Handler Record

Figure A.22, below, illustrates sample output for this command. (The values shown are those that could be expected if the record were added using the command shown in Figure A.17.)

In this specific instance, the user ID contained in **encrypted.file** (from Figure A.17) is **sparkie**, and the record was added by the user account with an ID of **sbuser**.

```
UNV3659I Connecting to local broker on port 7887.
UNV3406I Universal Event Monitor Server component 1117035117 started.
UNV3666I Load request started at 11:32:45 AM 05/25/2005.

Event Handler(s):
=================
Handler ID.................: dirlist
Max Acceptable Return Code.: 0
User ID....................: sparkie
Command....................: ls -al
Last Modified On...........: 05/25/2005 11:32:06 AM
Last Modified By...........: sbuser

UNV3667I Universal Event Monitor Load is ending successfully with exit code 0.
```

Figure A.33  Sample List Output

# A.6.6  Listing Multiple Event Definitions and Event Handlers Using Wildcards

In this example, the wildcards supported by `uemload` are demonstrated.

Wildcards can be used to select event definitions and event handlers whose respective IDs match the specified pattern.

- Asterisk ( **\*** ) can be used to match 0 or more characters.
- Question mark ( **?** ) can be used to match any single character.

All event definitions whose IDs start with the characters `event` are returned by the command below. In addition, all event handlers whose IDs begin with `handler0` and end with any two characters are selected.

```
uemload -list -event_id event* -handler_id handler0??
```

Figure A.34  Using Wildcards to List Records

## A.6.7  Add Record(s) Using a Definition File

In this example, a text file named **uemadd.txt** is used to add one or more records to the UEM databases. The contents of the file resemble those shown in Figure A.27.

A definition file allows multiple records to be added to the event definition and/or event handler databases at the same time. When no definition file is used, only a single record can be added to the database(s).

```
uemload –add –deffile uemadd.txt
```

Figure A.35  Add Database Record(s) Using a Definition File

# A.6.8 Add Record(s) Remotely, Using a Definition File Redirected from STDIN

In this example, a definition load file named `uemadd.txt` is used to add one or more records to the databases of a remote UEM Server. The contents of the file resemble those shown in Figure A.27.

Universal Command is used to execute UEMLoad on the remote UEM Server's system. The definition load file is redirected from standard input (stdin), which eliminates the step of copying the load file to the remote system before executing UEMLoad.

```
ucmd –cmd "uemload –add" –host rmthost –x rmtacctinfo.enc <uemadd.txt
```

Figure A.36  Redirect Definition File from stdin

## A.6.9  Add Record(s) Remotely, Using a Definition File Redirected from STDIN (for z/OS)

In this example, a definition load file named **MY.UEM.DATA(UEMDEF)** is used to add one or more records to the databases of a remote UEM Server. The contents of the file resemble those shown in Figure A.27.

Universal Command is used to execute UEMLoad on the remote UEM Server's system. It redirects standard input (stdin) from a data set allocated to the UNVIN ddname. This eliminates the step of copying the data set to the remote system before executing UEMLoad.

```
//STEP1     EXEC UCMDPRC
//UNVIN     DD  DISP=SHR,DSN=MY.UEM.DATA(UEMDEF)
//SYSIN     DD  *
 -HOST              dallas
 -USERID            joe
 -PWD               ahzidaeh
 -CMD               "/opt/universal/bin/uemload -add"
/*
```

Figure A.37  Redirect Definition File from STDIN (for z/OS)

# A.6.10  Definition File Format

The format of the definition load file for events and event handlers follows the keyword / value-pair convention used for Universal Products configuration files. However, because more than one definition can be specified in a load file, some additional conventions are used.

The **begin_event** and **end_event** keywords are used to mark the beginning and end, respectively, of each event definition entry.

The **begin_handler** and **end_handler** keywords are used to mark the beginning and end, respectively, of each event handler entry.

The **begin_script** and **end_script** keywords are used to mark the beginning and end, respectively, of any user script contained in the definition load file.

Lines that belong to the script must begin with the **stmt** keyword. Long **stmt** values that have to be split across lines can be done so using the **+** and **–** line continuation characters (as described in Section 2.2.6 Configuration File Syntax). These lines will be accepted verbatim, and no script syntax validation will be done. Lines will continue to be added to the script until an **end_script**, **end_handler**, **begin_handler**, or **begin_event** keyword is read, or the end of the file is reached.

If a parameter's value contains spaces, it must be enclosed in single ( **'** ) or double ( **"** ) quotation marks.

If quotes are to be saved as part of the parameter's value, enclose the value in single ( **'** ) quotation marks quotes, and use a set of double ( **"** ) quotation marks to enclose the quoted value (for example, **optname 'optval1 "optval2 optval2a" optval3'**).

The **script** keyword can be used in lieu of a **begin_script**/**end_script** block, in which case the contents of the specified file will be written to the event handler.

A sample definition file for Windows is shown in Figure A.27.

```
# Indented lines are for illustration only.  Leading spaces are
# ignored by UEMLoad.  Defaults will be used for any omitted
# values.

# Start of parameters for an event definition with an ID of
# "unix_event_sample".

begin_event
   event_id unix_event_sample
   event_type FILE
   comp_name uems
```

```
      state enable
      inact_date_time 2004.12.31,23:59
      triggered_id unix_script_sample
      filespec 'uem*.dat'
      rename_file yes
      rename_filespec '$(compname).$(compid).$(date).$(seqno)'
end_event

# End of parameters for event definition "unix_event_sample".

# Start of parameters for an event handler with an ID of
# "unix_script_sample".

begin_handler
      handler_id unix_script_sample
      handler_type SCRIPT
      maxrc 0
      userid uemuser
      begin_script
          stmt "#!/bin/sh"
          stmt ""
          stmt 'ls -al "/home/uem user"'
      end_script
      script_type bat
end_handler

# End of parameters for event handler "unix_script_sample".

# Start of parameters for an event definition with an ID of
# "unix_cmd_sample".

begin_handler
      handler_id unix_cmd_sample
      maxrc 0
      userid uemuser
      cmd '/home/uem user/someapp'
end_handler

# End of parameters for event definition "unix_cmd_sample".
```

Figure A.38  Definition File Sample - Windows

# Appendix B
# Standard I/O Redirection and Event Handler Processes

## B.1  Overview

Universal Event Monitor currently does not provide any native support for redirecting standard input and output to or from a process. However, it is possible to instruct the operating system to redirect standard I/O during process execution. I/O redirection is done using one of the following redirection symbols:

- **<** - Redirect standard input using an existing file. If an application supports it, standard input redirection can be used as a batch alternative to input that is normally obtained interactively from the user.

- **>** - Redirect standard output to a specified file. When a single **>** is used and the specified file exists, its contents are overwritten. The file is created if it does not exist.

- **>>** - Redirect standard output and append it to a specified file. If the specified file exists when this symbol is used, output generated by the process is added to the end of the specified file, preserving the file's current contents. The file is created if it does not exist.

- **2>** - Redirect standard error to a specified file. If the specified file exists, its contents are replaced. The file is created if it does not exist. If standard output also is being redirected to a file, an ampersand ( **&** ) followed by a **1** (which the operating system recognizes as an identifier for standard output) can be used in place of a file name. This causes standard error to be written to the same file used for standard output. For example, **2>&1** instructs the operating system to send all messages generated by the application that are destined for standard output and standard error to be written to the same file.

- **2>>** - Redirect standard error and append it to a specified file. If the specified file exists when this symbol is used, messages written by the application to standard error are added to the end of the specified file. If standard output also is being redirected to a file, an ampersand ( **&** ) followed by a **1** (which the operating system recognizes as an identifier for standard output) may be used in place of a file name. This causes standard error to be appended to the same file used for standard output.

For example, **2>>&1** instructs the operating system to send all messages generated by the application that are destined for standard output and standard error to be written to the same file.

This section describes how to redirect standard I/O for event handlers that execute a command or a script. It also explains how an event definition's *HANDLER_OPTIONS* option can be used to control standard i/o redirection.

# B.2  Command Execution

If an event handler executes a command, standard I/O redirection can be specified with the handler's *USER_COMMAND* option. The value of this option would be the same as it would be if you were issuing the command and redirecting I/O from the command prompt.

Figure B.1, below, provides examples.

```
Example 1
-cmd "dir >dirout.txt"

Example 2
-cmd "someapp <appinput.dat >applog.txt 2>&1"

Example 3
-cmd "dailybkup >>backup.log 2>>backup_err.log"
```

Figure B.1  Standard I/O redirection using the USER_COMMAND parameter.

Example 1 shows how to send the results of a `dir`ectory listing on a Windows system to a file in the current directory named `dirout.txt`.

Example 2 can be run on a Windows or UNIX system. It demonstrates the execution of an application named `someapp` that receives its input from a file named `appinput.dat`, and is redirecting all messages destined for `stdout` and `stderr` to a file named `applog.txt`. The contents of `applog.txt` are overwritten each time that the command executes.

Example 3 also can be run on a Windows or UNIX system. It executes an application named `dailybkup`, and appends all messages destined for `stdout` to `backup.log` and all messages destined for `stderr` to a file named `backup_err.log`.

Because the file name does not change with each invocation of the event handler, this method is best suited for event handler processes whose output does not need to be captured in unique files. If multiple event occurrences cause this event handler process to be executed at the same time, the output from each instance of the event handler process will be interleaved within the same file.

**Windows**

Special consideration should be given when executing several instances of an event handler process that redirects output to a single file. Sharing violations have been observed in this situation that prevent output from being properly captured. For these cases, it is recommended that a method be used that provides for assignment of a unique file name to each process instance.

# B.3  Script Execution

If an event handler executes a script, standard I/O redirection can be done within the script itself. This method provides a little more flexibility with respect to assigning unique names to files that store captured output. If unique names are needed, logic can be set up within the script to construct the name and to ensure that the file does not already exist.

Figure B.2 and Figure B.3, below, illustrate sample scripts that contain logic for generating unique file names.

<div style="background-color: orange; color: white; font-weight: bold; padding: 4px;">UNIX</div>

This sample is a Bourne shell script that uses a combination of environment variables set by the UEM Server, the script's process ID, and the current system date/time in order to construct a unique file name used to store the output of the **ls -al** command.

```sh
#!/bin/sh

rc=0

## Set a date string in the format yyyy.mm.dd.
dt=`date +%Y.%m.%d`

## Set a time string in the format hh.mm.ss
tm=`date +%H.%M.%S`

## Save the process ID
pid=$$

## Remove the extension from the file tracked by UEM.
uemFName=`basename "$UEMORIGFILE" | sed 's/\(^.*\)\.\(.*$\)/\1/'`

## Construct a filename using the name of the file tracked by UEM, the
## current date, the current time, and the process ID.
fname=$uemFName.$dt.$tm.$pid.txt

## Execute the command. Redirect the output to the file name
## constructed above.
ls -al >$fname

## Set the return code and exit the script
rc=$?
exit $rc
```

Figure B.2  Logic to Generate Unique File Name in Bourne Shell Script

This sample is a Windows batch file that uses a combination of environment variables set by the UEM Server and the current system date/time in an attempt to construct a unique file name used to store the output of the `dir` command. Because multiple instances of this script can be executing at the same time, and a process ID is not available to a Windows batch file, a loop exists to retry the file name construction if a matching file is found.

```
@echo off

:: Main script flow
set rc=0

call :SetVariables
call :SetupOutputFile

:: Now, execute the command. Redirect the contents of this directory
:: and all subdirectories to the file identified by %fname%.

dir /o/-p/s >%fname%
set rc=%ERRORLEVEL%

:: Exit the script
goto Exit


::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:: Subroutines
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

:SetVariables
:: Set a date string in the format yyyy.mm.dd.
for /f "tokens=2,3,4 delims=/ " %%a in ('date /t') do set dt=%%c.%%a.%%b

:: Set a time string in the format hh.mm.ss
for /f "tokens=1,2 delims=: " %%a in ('time /t') do set tm=%%a.%%b

:: Remove file extension
for /f "tokens=1,2 delims=." %%a in ("%UEMORIGFILE%") do set uemFName=%%a

:: Set the file name
set fname=%uemFName%.%dt%.%tm%.txt

:SetVariablesExit
goto :EOF

(continued)
```

```
(continued)

::
:: Test for the existence of the output file. If the file is found, loop
:: for a while then reset the time variable and check again.
::
:SetupOutputFile
:: If the file doesn't exist, create it to prevent another instance of this
:: script from grabbing it.
if not exist %fname% (
echo. >%fname%
goto SetupOutputFileExit
)

:: Otherwise, wait for a count of 10,000 (approx 1 sec) and try again.
if exist %fname% for /l %%I in (1,1,10000) do echo. >NUL

call :SetVariables
goto SetupOutputFile

:SetupOutputFileExit
goto :EOF

:Exit
exit %rc%
```

Figure B.3  Logic to generate unique file name in Windows batch file.

Note:   In both of these examples, the redirection is done at the point at which the
process is invoked. If the script itself also generates output that needs to be
captured, this file would need to be provided via the event definition's handler
options (see Section B.4 Handler Options).

# B.4 Handler Options

An event's HANDLER_OPTIONS option can be used to establish standard I/O redirection for every event handler process executed for the event. These options are specified using the HANDLER_OPTIONS option of the event definition or the `-handler_opts` command line option of the UEM Manager. In both implementations, the value specified for this parameter is appended to the command line that the UEM Server constructs to execute the specified user command or script.

This option has the same issues regarding unique file names as those described above in the Command Execution section. If it is possible that multiple, concurrently tracked event occurrences will result in the simultaneous execution of the event's handlers, all output generated by all handlers will be written to the same file. If output from each handler process needs to be captured in unique files, the method described in the Script Execution section above is a better alternative.

One advantage to using an event's handler options parameter is that an event's triggered, rejected, and/or expired event handlers can be changed without changing the way output is captured by the respective event handler process.

An event's handler options parameter may also be used to redirect standard input from a file. Capturing output may then be handled with the handler options parameter or by using one of the other methods described above.

The examples in Figure B.4, below, demonstrate the use of this parameter for event definitions and event handlers added with the UEMLoad utility, and for events defined and executed from a UEM Manager's command line.

```
Example 1
UEMLoad utility:
uemload -add -event_id event001 -handler_opts ">netstat.txt"
-triggered_id handler001 -expired_id expiredhandler
-event_type file -filespec uemtest.dat
uemload -add -handler_id handler001 -cmd netstat
-encryptedfile userfile.enc
uemload -add -handler_id expiredhandler -cmd "echo Event expired"
-encryptedfile userfile.enc

UEM Manager command line:
uem -event_type file -filespec uemtest.dat
-host rmthost -user rmtuser -pwd password
-handler_opts ">dirout.txt" -triggered -cmd netstat
-expired -cmd "echo Event expired"

Example 2
UEMLoad utility:
uemload -add -event_id event002 -triggered_id handler002
-handler_opts "<appinput.dat >applog.txt 2>&1"
-event_type file -filespec uemtest.dat
uemload -add -handler_id handler002 -cmd someapp
-encryptedfile userfile.enc

UEM Manager command line:
uem -event_type file -filespec uemtest.dat
-host rmthost -user rmtuser -pwd password
-handler_opts "<appinput.dat >applog.txt 2>&1"
-triggered -cmd someapp

Example 3
UEMLoad utility:
uemload -add -event_id event003 -inact_date_time 2005.12.31,23:59
-handler_opts ">>backup.log 2>>backup_err.log"
-rejected_id handler003 -event_type file -filespec uemtest.dat
uemload -add -handler_id handler003 -cmd dailybkup

UEM Manager command line:
uem -event_type file -inact_date_time 2005.12.31,23:59 -filespec uemtest.dat
-host rmthost -user rmtuser -pwd password
-handler_opts ">>backup.log 2>>backup_err.log"
-rejected -cmd dailybkup
```

Figure B.4  Using handler options when storing event definitions

For each of the event definitions shown above, the OS installed on the host `rmthost` can be either Windows or UNIX. If the UEM Manager is executed from z/OS, the name of the executable (that is, `uem`) would be omitted. However, the rest of the parameters would be entered as shown in the JCL's SYSIN DD statement.

The first example shows an event of type *FILE* that will monitor a file named `uemtest.dat`. If an occurrence of this file is detected, the `netstat` command will be executed when the occurrence enters a triggered state. If the event expires, an "Event expired" message is issued. In both situations, the output from the command is written to a file named `netstat.txt`.

The second example also shows an event of type *FILE* that is set up to monitor a file named `uemtest.dat`. If an occurrence of this file is triggered, the application `someapp` will be executed. According to the `-handler_options` parameter, this application will receive input redirected from a file named `appinput.dat`. All output, including output destined for standard error, is written to a file named `applog.txt`.

Finally, the last example is also monitors a file named `uemtest.dat`. This event executes an event handler process only if an occurrence of the file is detected and that file fails to complete by *midnight* on *31 December 2005*. The handler process executed is named `dailybkup`. All messages issued by the application that are destined for standard output are written at the end of a file named `backup.log`. All messages destined for standard error are appended to a file named `backup_err.log`. One or both files are created if they do not exist when the handler process is executed.

# Appendix C
# Environment Variables Set by Universal Event Monitor

Before it executes an event handler process, Universal Event Monitor can set one or more environment variables that contain information regarding a monitored event and/or an event occurrence. The environment variables that are set depend on a monitored event's type and processing state.

The environment variables set for the different event types are listed in Table C.1, below. The three right-most columns represent the event processing states:

- (**T**)riggered
- (**R**)ejected
- (**E**)xpired

| Event Type | Variable Name | Description | T | R | E |
|---|---|---|---|---|---|
| FILE | UEMFILESPEC | File specification from event definition | √ | √ | √ |
| FILE | UEMORIGFILE * | Name of file monitored and tracked by UEM, including the complete path | √ | √ | |
| FILE | UEMORIGFNAME | Name of file monitored and tracked by UEM | √ | √ | |
| FILE | UEMORIGPATH | Location of file monitored and tracked by UEM | √ | √ | |
| FILE | UEMRENAMEDFILE * | Name of file after being renamed by UEM, including the complete path | √ | | |
| FILE | UEMRENAMEDFNAME | Name of file after being renamed by UEM | √ | | |
| FILE | UEMRENAMEDPATH | Location of file after being renamed by UEM | √ | | |
| FILE | UEMFILESIZE | Last recorded file size in space unit of bytes | √ | √ | |

Table C.1  Environment Variables Set by Universal Event Monitor

# Appendix D
# Customer Support

Stonebranch, Inc. provides customer support, via telephone and e-mail, for Universal Event Monitor and all Universal Products.

## TELEPHONE

Customer support via telephone is available 24 hours per day, 7 days per week.

**North America**

**(+1) 678 366-7887, extension 6**

**(+1) 877 366-7887, extension 6  [toll-free]**

**Europe**

**+49 (0) 700 5566 7887**

## E-MAIL

**All Locations**

**support@stonebranch.com**

Customer support contact via e-mail also can be made via the Stonebranch website:

**www.stonebranch.com**

**stonebranch**

950 North Point Parkway, Suite 200

Alpharetta, Georgia 30005

U.S.A.