



**Infitran 5.1.0**

**User Guide**

© 2013 by Stonebranch, Inc. All Rights Reserved.

1. Infitran 5.1.0 User Guide	5
1.1 Infitran - Overview	6
1.2 Infitran - Transferring Files to and from Remote Systems	11
1.2.1 Transferring Files - Overview	12
1.2.2 Transfer Sessions	13
1.2.3 Transferring Files - Examples	15
1.2.3.1 Copy a File to an Existing zOS Sequential Data Set	16
1.2.3.2 Copy a File to a New zOS Sequential Data Set	18
1.2.3.3 Copy a zOS Sequential Data Set to a File	19
1.2.3.4 Copy a Set of Files to an Existing zOS Partitioned Data Set	20
1.2.3.5 Copy a Set of Files to a New zOS Partitioned Data Set	22
1.2.3.6 Simple File Copy to the Manager - Windows and UNIX	23
1.2.3.7 Simple File Copy to the Server - Windows and UNIX	24
1.2.3.8 Copy a Set of Files - Windows and UNIX	25
1.2.3.9 Copy a File to an Existing IBM i File	26
1.2.3.10 Copy an IBM i Data Physical File to a File	27
1.2.3.11 Copy a Set of Files to an Existing Data Physical File	28
1.2.3.12 Copy a File to a New IBM i Data Physical File	29
1.2.3.13 Copy a File to a New IBM i Source Physical File	30
1.2.3.14 Copy a Set of Files to a New Data Physical File on IBM i	31
1.2.3.15 Copy Different Types of IBM i Files Using forfiles and \$(_file.type)	32
1.2.3.16 Invoke a Script from an IBM i Batch Job	33
1.3 Infitran - Remote Execution	34
1.3.1 Remote Execution - Overview	35
1.3.2 Remote Execution - Primer	36
1.3.3 Remote Execution - Examples	39
1.3.3.1 Windows Directory Listing Using a Batch File - Default Directory	40
1.3.3.2 Windows Directory Listing Using a Batch File - Returned File	42
1.3.3.3 UNIX - Listing Using a Shell Script	44
1.3.3.4 UNIX - Integrating UDM with FTP Using a Shell Script	46
1.3.3.5 UNIX - Integrating UDM with FTP Using a Command Reference	48
1.3.3.6 IBM i from Windows, UNIX, or IBM i - exec Command Return Codes	50
1.4 Infitran - Remote Execution for SAP Systems	52
1.4.1 Remote Execution for SAP Systems - Overview	53
1.4.2 Remote Execution for SAP Systems - Examples	54
1.4.2.1 Raising an SAP Event for zOS Example	55
1.4.2.2 Raising an SAP Event for UNIX Example	57
1.5 Infitran - Web Services Execution	59
1.5.1 Web Services Execution - Overview	60
1.5.2 Web Services Execution - Examples	61
1.5.2.1 Inbound JMS Implementation	62
1.5.2.2 Inbound SOAP Implementation	66
1.6 Infitran - Event Monitoring and File Triggering	72
1.6.1 Event Monitoring and File Triggering - Overview	73
1.6.2 Event Monitoring and File Triggering - Universal Event Monitor	74
1.6.2.1 Event Monitoring and File Triggering - Universal Event Monitor - Overview	75
1.6.2.2 Storing Event Definitions and Event Handlers	77
1.6.2.3 Universal Event Monitor - Monitoring a Single Event	79
1.6.2.4 Universal Event Monitor - Monitoring Multiple Events	81
1.6.3 Event Monitoring and File Triggering - UEMLoad	83
1.6.4 Event Monitoring and File Triggering - Examples	85
1.6.4.1 Starting an Event-Driven UEM Server - zOS	86
1.6.4.2 Refreshing an Event-Driven UEM Server - zOS	87
1.6.4.3 Using a Stored Event Handler Record - zOS	88
1.6.4.4 Handling an Event with a Script - zOS	90
1.6.4.5 Handling an Expired Event - zOS	92
1.6.4.6 Continuation Character ( - ) in zOS Handler Script	94
1.6.4.7 Continuation Character ( + ) in zOS Handler Script	95
1.6.4.8 Continuation Characters ( - and + ) in zOS Handler Script	96
1.6.4.9 Using a Stored Event Handler Record - Windows	97
1.6.4.10 Execute Script for Triggered Event Occurrence - Windows	99
1.6.4.11 Handling an Expired Event - Windows	101
1.6.4.12 Add a Single Event Record - Windows	103
1.6.4.13 Add a Single Event Handler Record - Windows	104
1.6.4.14 List All Event Definitions - Windows	105
1.6.4.15 Export Event Definition and Handler Databases - Windows	106
1.6.4.16 List a Single Event Handler Record - Windows	107
1.6.4.17 List Event Definitions and Handlers Using Wildcards - Windows	108
1.6.4.18 Add Record(s) Using Definition File - Windows	109
1.6.4.19 Add Records Remotely Redirected from STDIN - Windows	110
1.6.4.20 Add Records Redirected from STDIN (for zOS) - Windows	111
1.6.4.21 Definition File Format - Windows	112
1.6.4.22 Using a Stored Event Handler Record - UNIX	115
1.6.4.23 Execute Script for Triggered Event Occurrence - UNIX	117
1.6.4.24 Handling an Expired Event - UNIX	119

1.6.4.25 Add a Single Event Record - UNIX	121
1.6.4.26 Add a Single Event Handler Record - UNIX	122
1.6.4.27 List All Event Definitions for UNIX	123
1.6.4.28 List a Single Event Handler Record for UNIX	124
1.6.4.29 Export Event Definition and Handler Databases - UNIX	125
1.6.4.30 List Event Definitions and Handlers Using Wildcards for UNIX	126
1.6.4.31 Add Record(s) Using Definition File for UNIX	127
1.6.4.32 Add Record(s) Remotely Redirected from STDIN - UNIX	128
1.6.4.33 Add Record(s) Remotely Redirected from STDIN (for zOS) - UNIX	129
1.6.4.34 Definition File Format - UNIX	130
1.7 Infitran - Encryption	133
1.7.1 Encryption - Overview	134
1.7.2 Encryption - Examples	136
1.7.2.1 Creating Encrypted Command File for zOS	137
1.7.2.2 Using Encrypted Command File on zOS	139
1.7.2.3 Creating Encrypted Command File for Windows	140
1.7.2.4 Using Encrypted Command File on Windows	142
1.7.2.5 Creating Encrypted Command File for UNIX	143
1.7.2.6 Using Encrypted Command File on UNIX	145
1.7.2.7 Creating Encrypted Command File for IBM i	146
1.7.2.8 Using Encrypted Command File on IBM i	148
1.8 Infitran - Configuration Management	149
1.8.1 Configuration Management - Overview	150
1.8.2 Configuration Methods	151
1.8.2.1 Configuration Methods - Overview	152
1.8.2.2 Configuration Methods - Command Line	153
1.8.2.3 Configuration Methods - Command File	155
1.8.2.4 Configuration Methods - Environment Variables	156
1.8.2.5 Configuration Methods - Configuration File	158
1.8.3 Configuration Options	160
1.8.4 Remote Configuration	161
1.8.5 Universal Configuration Manager	165
1.8.5.1 Universal Configuration Manager - Usage	166
1.8.5.2 Universal Configuration Manager - Installed Components	170
1.8.6 Configuration Refresh	180
1.8.7 Refreshing via Universal Control Examples	182
1.8.7.1 Refreshing Universal Broker from zOS	183
1.8.7.2 Refreshing a Component from zOS	185
1.8.7.3 Refreshing Universal Broker from Windows	186
1.8.7.4 Refreshing a Component from Windows	187
1.8.7.5 Refreshing Universal Broker from UNIX	188
1.8.7.6 Refreshing a Component from UNIX	189
1.8.7.7 Refreshing Universal Broker from IBM i	190
1.8.7.8 Refreshing a Component from IBM i	191
1.8.8 Merging Configuration Options Examples	192
1.8.8.1 Files Used in UPI Merge Examples	193
1.8.8.2 Merge Configuration Files Using Program Defaults	195
1.8.8.3 Merge Configuration Files Introducing New Options	197
1.8.8.4 Merge Configuration Files Using Installation-Dependent Values	199
1.9 Infitran - Component Management	201
1.9.1 Component Management - Overview	202
1.9.2 Component Definition	203
1.9.3 Component Definition Options	204
1.9.4 Starting and Stopping Components	205
1.9.5 Starting and Stopping Components - Examples	206
1.9.5.1 Starting and Stopping Universal Broker for zOS	207
1.9.5.2 Starting Universal Broker for Windows	208
1.9.5.3 Starting Universal Broker for UNIX	210
1.9.5.4 Starting, Ending, and Working with Universal Broker for IBM i	212
1.9.5.5 Starting and Stopping Universal Enterprise Controller for zOS	214
1.9.5.6 Starting and Stopping Universal Enterprise Controller for Windows	216
1.9.5.7 Starting a zOS Component via Universal Control	217
1.9.5.8 Stopping a zOS Component via Universal Control	218
1.9.5.9 Starting a Windows Component via Universal Control	219
1.9.5.10 Stopping a Windows Component via Universal Control	220
1.9.5.11 Starting a UNIX Component via Universal Control	221
1.9.5.12 Stopping a UNIX Component via Universal Control	222
1.9.5.13 Starting an IBM i Component via Universal Control	223
1.9.5.14 Stopping an IBM i Component via Universal Control	224
1.9.6 Maintaining Universal Broker Definitions in UEC Database	225
1.9.6.1 Maintaining Broker Definitions in UEC Database - zOS and Windows	226
1.9.6.2 Maintaining Broker Definitions in UEC Database - zOS	231
1.9.6.3 Maintaining Broker Definitions in UEC Database - Windows	233
1.10 Infitran - Messaging and Auditing	235
1.10.1 Messaging and Auditing - Overview	236

1.10.2 Messaging	237
1.10.3 Auditing	240
1.10.4 Creating Write-to-Operator Messages - Examples	241
1.10.4.1 Issue WTO Message to zOS Console	242
1.10.4.2 Issue WTO Message to zOS Console and Wait for Reply	243
1.11 Infitran - Message Translation	244
1.11.1 Message Translation - Overview	245
1.11.2 Message Translation - Examples	247
1.11.2.1 Translating Error Messages	248
1.11.2.2 Execute Universal Message Translator from zOS	250
1.11.2.3 Execute Universal Message Translator from Windows	251
1.11.2.4 Execute Universal Message Translator from UNIX	252
1.11.2.5 Execute Universal Message Translator from IBM i	253
1.12 Infitran - Monitoring and Alerting	254
1.12.1 Monitoring and Alerting - Overview	255
1.12.2 Querying for Job Status and Activity - Examples	256
1.12.2.1 Querying - Universal Query Output	257
1.12.2.2 Querying - Universal Query for zOS	258
1.12.2.3 Querying - Universal Query for UNIX and Windows	259
1.12.2.4 Querying - Universal Query for IBM i	260
1.13 Infitran - Windows Event Log Dump	261
1.13.1 Windows Event Log Dump - Overview	262
1.13.2 Windows Event Log Dump - Examples	263
1.13.2.1 Execute Universal Event Log Dump from a Windows Server	264
1.14 Infitran - Fault Tolerance Implementation	265
1.15 Infitran - Network Data Transmission	268
1.15.1 Network Data Transmission - Overview	269
1.15.2 SSL (Secure Socket Layer) Protocol	270
1.15.3 Universal V2 Protocol	272
1.15.4 Universal Application Protocol	273
1.15.5 Network Data Transmission - Configurable Options	275
1.16 Infitran - zOS CANCEL Command Support	278
1.16.1 zOS CANCEL Command Support - Overview	279
1.16.2 zOS CANCEL Command Support - Universal Data Mover	280
1.17 Infitran - Glossary	281

# Infitran 5.1.0 User Guide

## Infitran - Overview

- [What is Infitran?](#)
- [What Can Infitran Do for Me?](#)
- [Infitran Features](#)
- [Infitran Components](#)
  - [Universal Data Mover](#)
  - [Universal Event Monitor](#)
  - [Universal Event Monitor for SOA](#)
  - [Universal Enterprise Controller](#)
  - [Universal Enterprise Controller Client Applications](#)
  - [Universal Broker](#)
  - [Universal Automation Center Agent](#)
  - [Workload Automation 5 Utilities](#)
- [Infitran Limited Use Components](#)
  - [Universal Command](#)
  - [Universal Command Agent for SOA](#)
  - [Universal Connector](#)

## What is Infitran?

Infitran (Intelligent File Transfer) is the Stonebranch Inc. business solution for Managed File Transfer.

In addition to the basic features inherent in the managed file transfer of files between servers and applications – security, visibility, manageability, reliability, and compliance – Infitran provides additional features for intelligent file transfer.

Infitran inter-operates with your current job scheduling and automation tools, providing complete visibility for all scheduled and automated event-driven file transfers; not only end-to-end from the file movement perspective, but also top-to-bottom integration with application processes.

Comprehensive and intuitive filtering in Infitran allows you to find information about file transfer activity such as failed transfers and successful transfers, how much data was transferred, and transfer attributes.

Infitran provides a layered approach to security enforcement that protects networks and controls access to data and servers. Data encryption can be enforced in a way that ensures compliance requirements are always met.

## What Can Infitran Do for Me?

The intelligent file transfer of data provided by Infitran lets you streamline business processes by optimizing the integration of file transfers with your business processes. This helps you avoid delays and maximize revenue.

Using Infitran enables you to securely transfer files to external partners without disruption of their current business processes. The integration capabilities and ease of use provided by Infitran enable you to manage thousands of servers with minimum interaction.

Intelligently transferred data supports your ability to analyze and plan. Infitran ensures that your Managed File Transfer environment runs effectively and efficiently, providing historical data to make informed decisions.

Infitran enables you to report on data related to all aspects of file transfers specific to user needs. Valuable data is preserved for compliance reporting. All file transfer events that are related are recorded in a central database that can be extracted for reporting and auditing purposes.

Infitran delivers flexible visibility tools and capabilities to meet your own business and operational needs for both internal and external communications. With its proactive monitoring, Infitran provides you with the maximum possible time to address any technical issues that may arise. You do not have to wait for a failed transfer to discover server or network problems.

## Infitran Features

The features that make Infitran an intelligent file transfer solution encompass a variety of core and supporting functionality.

The following text describes these features and provides links to detailed information about each one in this document. This includes examples that illustrate feature implementation and links to detailed technical information about the [Infitran Components](#) used in that implementation.

The core feature of Infitran is [Transferring Files to and from Remote Systems](#) in a manner that is both secure and efficient. Transfer sessions can be initiated between the machine initiating the transfer and a remote machine, or between two remote machines.

Elaborate [Event Monitoring and File Triggering](#) functionality enables the monitoring local and remote system events, and permits execution of

system commands or scripts based on the outcome of the events.

[Web Services Execution](#) enables Infitran to create file-based events from inbound Internet and message-based application messages, and then write the events to file, thus integrating those applications with Infitran system management.

For Infitran systems on Windows, the [Windows Event Log Dump](#) feature offers the ability to select records from a Windows event log and write them to a specified output file.

Infitran's array of [Databases](#) record information throughout an enterprise. Information on all Infitran installations, including the current status of every component is maintained, as well as user and configuration data, is maintained. The databases also store information that defines Infitran system occurrences (events), the action to implement for those events, and the progress of each event.

The [Monitoring and Alerting](#) feature of Infitran provides for monitoring the status and activity of all Infitran Agents in an enterprise and the posting of alerts regarding the statuses. This information is available through a user interface, but it also provides for the command line querying of a job status and activity of a specific Agent.

[Configuration Management](#) tools allow for flexible methods of configuration. [Remote Configuration](#) enables all systems in an enterprise to be configured from a single machine. On Windows systems, configuration can be made via Infitran's [Universal Configuration Manager](#) graphical user interface.

Additionally, Infitran offers various methods for the [Configuration Refresh](#) of all component data. Infitran [Component Management](#) is built around the particular needs of individual components.

A rich [Messaging and Auditing](#) system provides continuous system feedback via six different levels of messages. The system can be modified to provide different levels of messaging, from diagnostic and alert messages, which are always provided, to audit level, which produces messaging on all aspects of system functionality.

With [Message Translation](#), error messages returned by commands can be translated into return codes.

Infitran [Security](#) is enabled at many levels. Access to files, directories, configuration data is strictly controlled, as is user authentication. All Infitran components implement [Network Data Transmission](#) using the TCP/IP protocol. For [Encryption](#) of transmitted data, Infitran uses SSL to provide the highest level of security available.

[Fault Tolerance Implementation](#) allows Infitran to recover from an array of error conditions at the network level, such as may occur in any large enterprise. Since network fault tolerance enables servers to continue processing even after a job is canceled, Infitran's [zOS Cancel Command Support](#) allows – on z/OS operating systems – termination of those jobs.

Infitran's [Remote Execution](#) permits the execution of system commands on remote machines. Additionally, [Remote Execution for SAP Systems](#) permits SAP events to be executed on remote SAP systems.

## Infitran Components

[Infitran Features](#) are implemented via a set of inter-related components that provide for a complete intelligent file transfer business solution.

One or more components provide the technical structure for the implementation of every feature.

### Universal Data Mover

[Universal Data Mover](#) is the core component for Infitran's managed file transfer functionality. In a secure and automated manner, it allows you to transfer data between any platforms in your environment and initiated from any platform.

In every Universal Data Mover transfer operation, a manager receives commands from the user through an interactive session and/or an external script file. It then establishes a transfer session, invoking primary and secondary servers, which actually conduct the transfer operation.

A transfer session either can be a two-party session, in which the manager also serves as the primary transfer server, or a three-party session, in which the manager acts solely as a control point for transfer operations, sending commands to the primary and secondary servers.

### Universal Event Monitor

[Universal Event Monitor](#) provides a platform-independent means of monitoring local and remote system events, and executing system commands and scripts based on the outcome of those events.

It integrates with your workload management infrastructure to initiate both movement of the data to the appropriate platform and immediate processing of the data as soon as it is available by executing system commands and scripts based on the outcome of the events that it is monitoring.

### Universal Event Monitor for SOA

[Universal Event Monitor for SOA](#) - the SOA "Listener" – integrates Internet and message-based applications with systems management functions, letting you create file-based events from inbound Internet and message-based messages, and write the events to file.

It integrates Internet and message-based applications with systems management functions such as alerting and notification, incident and problem management, Job scheduling, and data movement.

## Universal Enterprise Controller

[Universal Enterprise Controller](#) provides alerts for activity and availability of the Infitran components and Agents installed throughout your enterprise. It prevents jobs from starting and files from being transferred or processed during hardware failures or network issues.

Universal Enterprise Controller issues alerts when a component becomes unreachable or unavailable, as well as when the component is again available, and lets you route these alerts to your existing automation console.

Universal Enterprise Controller also provides the management layer that enables the Universal Event Subsystem and Universal Enterprise Client Applications (I-Activity Monitor, I-Management Console, and I-Administrator), to centralize visibility and management of your workload infrastructure.

## Universal Event Subsystem

The [Universal Event Subsystem](#) records, routes, and manages event messages generated by Infitran components. Event messages are generated whenever a component performs an action that impacts the computing environment on which it executes. The records are stored centrally and can be exported for audit and history reporting, as well as for archival.

## Universal Enterprise Controller Client Applications

[Universal Enterprise Controller Client Applications](#) are a suite of three stand-alone client applications for Windows operating systems used to manage and provide visibility to the Infitran infrastructure:

### I-Activity Monitor

I-Activity Monitor provides end-to-end visibility of workload management activity throughout your Infitran environment via a graphical user interface that displays information about the current status and posted alerts for all Agents and SAP systems being monitored by Universal Enterprise Controller.

### I-Management Console

The I-Management Console client application provides a graphical user interface for remote configuration of all Stonebranch Agents in an enterprise from a single machine. It also lets you define standard security access and authentication policies and ensure that they are active across all servers, as well as define which users are allowed to change the policies.

### I-Administrator

The I-Administrator client application lets you maintain information on all Agents that Universal Enterprise Controller monitors and the SAP systems to which Universal Enterprise Controller has access. It lets you add, modify, and delete users, Agents, groups, and SAP systems, as well as maintain Universal Enterprise Controller users and their permissions.

## Universal Broker

[Universal Broker](#), required on all systems running Infitran, manages Infitran components. It receives requests to start (or restart) a component on behalf of a user (person or component). Universal Broker tracks and reports on all components that it has started until their completion.

## Universal Automation Center Agent

The [Universal Automation Center Agent](#) (UAG) component provides agent services for an Automation Center server. UAG enables an Automation Center server to schedule workload, transfer files, and monitor events on the agent system. UAG completely and seamlessly integrates with Automation Center server to provide distributed, workload automation throughout the enterprise.

Universal Automation Center Agent (UAG) automatically starts when the Universal Broker starts and stops when the Universal Broker stops.

## Workload Automation 5 Utilities

[Workload Automation 5 Utilities](#), included as part of the Infitran business solution, perform a variety of functions for one or more operating systems.

## Universal Certificate



Infitran supports X.509 version 1 and version 3 certificates to securely identify users and computer systems. Although implementing a fully featured PKI infrastructure is beyond the scope of Infitran, if your organization has not yet established one, the Universal Certificate utility can be used to create digital certificates and private keys.

## Universal Control

Universal Control provides the ability to start and stop Infitran components, and to refresh component configuration data.

## Universal Database Dump

Universal Database Dump Berkeley db\_dump utility is tailored specifically for Stonebranch databases. It allows you to dump one or more databases for backup and restore purposes.

## Universal Database Load

Universal Database Load Berkeley db\_load utility is tailored specifically for Stonebranch databases. It provides the ability to restore a database that has been previously dumped.

## Universal Display Log File

Universal Display Log File, available for the IBM i operating system, provides the ability to read job log files, write them to standard out, and, optionally, delete the files after they are read.

## Universal Encrypt

Universal Encrypt encrypts the contents of command files into an unintelligible format (for privacy reasons).

## Universal Event Log Dump

Universal Event Log Dump (UELD) is a utility that selects records from one of the Windows event logs and writes them to a specified output file.

## Universal Message Translator

Universal Message Translator translates error messages into return (exit) codes based on a user-defined translation table.

## Universal Products Install Merge

The Universal Products Install Merge (UPIMERGE) utility merges options and values from one component configuration file or component definition file with another.

## Universal Query

Universal Query queries any Universal Broker for Broker-related and active component-related information. This utility can be issued from any Infitran installation to query any broker in the Stonebranch infrastructure.

## Universal Return Code

The Universal Return Code utility is a Windows utility that performs the function of ending a process with a return code that is equal to its command line argument.

## Universal Spool List

Universal Spool List provides the ability to list database records. The functions that Universal Spool List provide are required for possible database clean-up or problem resolution at the direction of Stonebranch, Inc. Customer Support.

## Universal Spool Remove

Universal Spool Remove provides the ability to remove component records from the Stonebranch databases. Universal Spool Remove should only be used at the direction of Stonebranch, Inc. Customer Support.

## Universal Submit Job

The Universal Submit Job (USBMJOB) utility is a command for the iSeries environment that encapsulates the IBM Submit Job (SBMJOB) command.

## Universal Write-to-Operator

The Universal WTO (UWTO) utility is a command line utility for the z/OS UNIX System Services (USS) environment. It issues two types of messages to z/OS consoles:

1. Write-To-Operator (WTO) messages
2. Write-To-Operator-with-Reply (WTOR) messages.

## Infitran Limited Use Components

The following Indesca business solution components are included in Infitran with limited use through the `exec` and `execsap` commands only.



### Note

For detailed information on these components, see the [Indesca 5.1.0 User Guide](#).

## Universal Command

[Universal Command](#), the core component for Indesca's enterprise scheduling functionality, allows you to extend the command line interface of a local operating system to the command line interface of any remote system that can be reached on a computer network.

A Universal Command Manager, on the local system, extends a command line interface to a remote system. A Universal Command Server, on the remote system, executes commands on behalf of the Manager. The Manager runs as long as the remote command runs. When the remote command ends, the Manager ends with the exit status of the remote command.

### Limitations of Use

If Universal Command is on the same system as [Universal Data Mover](#), you can execute system commands on remote machines using the Universal Data Mover `exec` command.

## Universal Command Agent for SOA

[Universal Command Agent for SOA](#) – the SOA "Publisher" – lets you extend the workload execution and management features of Indesca to Internet and message-based workload. It receives its payload input from Universal Command through STDIN. When the parameters and data are passed in, the workload execution request is processed and any return data is passed back to Universal Command.

Universal Command Agent for SOA can be initiated from a variety of sources, regardless of platform, enabling you to consolidate your Internet and message-based workload within your current enterprise scheduling environment.

### Limitations of Use

If Universal Command Agent for SOA and Universal Command are on the same system as [Universal Data Mover](#), you can execute system commands on remote machines using the Universal Data Mover `exec` command. The system commands can, in turn, execute Universal Command Agent for SOA workloads.

## Universal Connector

[Universal Connector for Use with SAP® ERP](#) is a command line interface that lets you manage SAP background processing tasks from any scheduling system on any platform.

Universal Connector provides the functionality to integrate SAP systems into both local administrative tools and enterprise system management infrastructures. It lets you extend your existing scheduling tools to SAP batch workloads, enabling you to manage all of your scheduling activities from one tool.

Certified by SAP, Universal Connector uses standard SAP interfaces only, such as XBP3.0.

### Limitations of Use

If Universal Connector is on the same system as [Universal Data Mover](#), you can execute SAP events using the Universal Data Mover `execsap` command.

## **Infitran - Transferring Files to and from Remote Systems**

## Transferring Files - Overview

- [Transferring Files to and from Remote Systems](#)
- [Transfer Operation Components](#)
  - [Manager](#)
  - [Primary Server](#)
  - [Secondary Server](#)

### Transferring Files to and from Remote Systems

Infitran's file transfer solution, developed specifically for corporate IT infrastructures and automated data center environments, makes transferring data between various enterprise and desktop platforms reliable and easy.

These pages describe the framework in which transfers are made, and provides examples of file transfers from all supported operating systems.

### Transfer Operation Components

There are three components to any Infitran transfer operation:

1. Manager
2. Primary server
3. Secondary server

The Manager can act as the primary server, depending on the type of transfer session: two-party or three-party (see [Transfer Sessions](#)).

The secondary server is always a separate and distinct component invoked via the Universal Broker.

#### Manager

The Universal Data Mover Manager processes commands using Universal Data Mover's scripting language. The Manager receives commands from the user through an interactive session, an external script file, or some combination of the two. Before the Manager can initiate any transfer operations, it must first establish a transfer session where it invokes the primary and secondary servers, which actually conduct the transfer operations.

#### Primary Server

When a transfer session is being established, the Universal Data Mover Manager invokes the primary server, which acts as the first endpoint in a transfer operation. In turn, the primary server invokes the secondary server, providing a single path of communication. The primary server also acts as a relay for the Manager, forwarding on any messages for the secondary server from the Manager. This single message pipeline reduces the number of connections needed for three-party transfers (see [Three-Party Transfer Sessions](#)).

#### Secondary Server

The secondary server acts as the second endpoint in a transfer operation. Data is transferred between primary and secondary servers, with either endpoint able to act as the source in a transfer operation.

## Transfer Sessions

- Overview
  - Logical Names
  - Two-Party Transfer Sessions
  - Three-Party Transfer Sessions
- Transfer Sessions (Illustrated)

### Overview

Transfer operations take place within the context of a transfer session. A transfer operation is initiated once the Universal Data Mover Manager has established a transfer session with the primary and secondary transfer servers (see [UDM Transfer Sessions](#)). All subsequent transfer operations take place between the primary and secondary transfer servers.

Universal Data Mover transfer sessions can be either two-party or three-party.

### Logical Names

When a transfer session is established, the user gives each server a unique logical name. Commands addressed to a particular server reference this logical name.

### Two-Party Transfer Sessions

For a two-party transfer session, the Universal Data Mover Manager also acts as the primary transfer server, running in the directory – and under the user ID – under which the Manager was launched. This means that the machine on which Manager resides is the first endpoint of the transfer.

With a two-party transfer session, the secondary server is invoked by the manager / primary server via the Universal Broker. The second endpoint of the transfer session will be on the machine in which the secondary server was spawned. Transfer operations occur between the manager / primary server and the secondary server.

(See the following figure.)

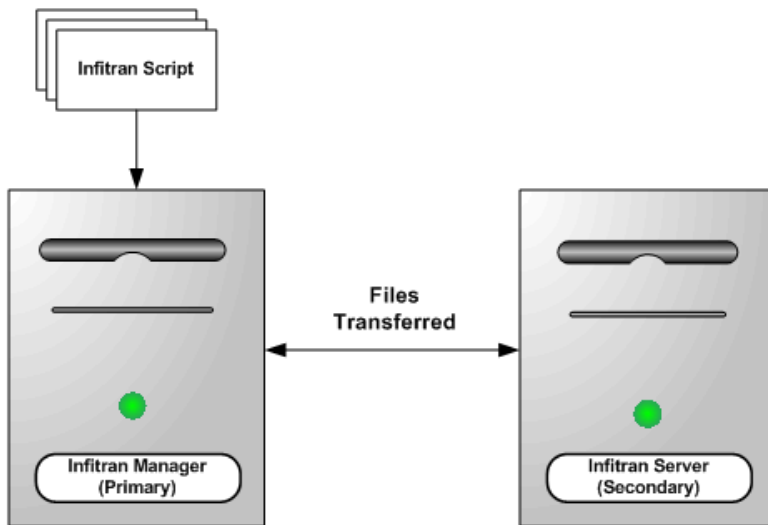
### Three-Party Transfer Sessions

For a three-party transfer session, the Universal Data Mover Manager acts solely as a control point for transfer operations, sending commands to the primary and secondary servers to be executed. Both the primary and secondary servers are spawned via the Universal Broker, and transfer operations take place between the two machines under which these servers are running.

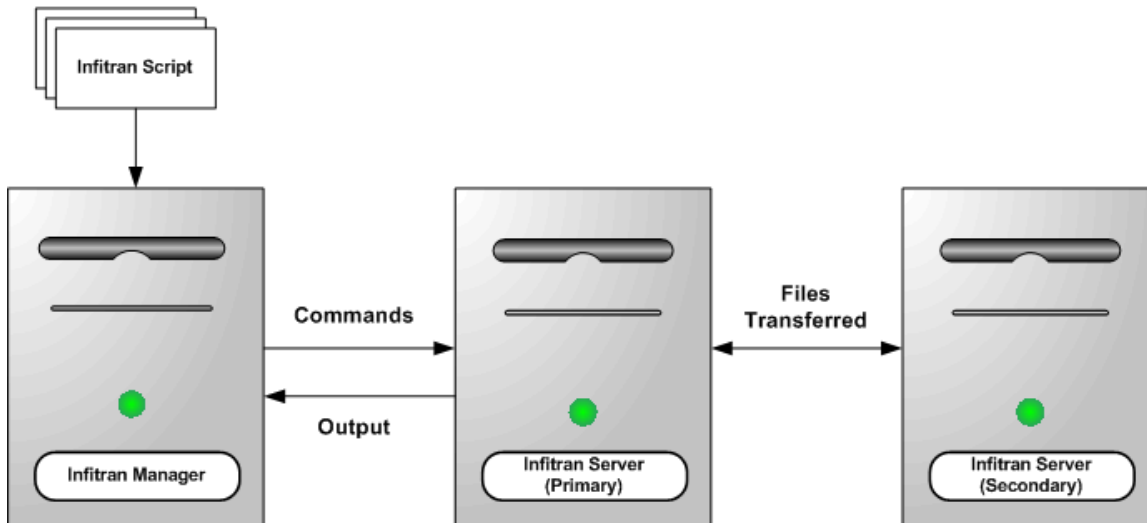
(See the following figure.)

### Transfer Sessions (Illustrated)

## Two-Party Transfer



## Three-Party Transfer



## Transferring Files - Examples

- [Transferring Files Examples - z/OS](#)
- [Transferring Files Examples - Windows and UNIX](#)
- [Transferring Files Examples - IBM i](#)

### Transferring Files Examples - z/OS

- [Copy a File to an Existing z/OS Sequential Data Set](#)
- [Copy a File to a New z/OS Sequential Data Set](#)
- [Copy a z/OS Sequential Data Set to a File](#)
- [Copy a Set of Files to an Existing z/OS Partitioned Data Set](#)
- [Copy a Set of Files to a New z/OS Partitioned Data Set](#)

These examples illustrate two-party transfer sessions between z/OS and UNIX. As appropriate for the example being illustrated, there are versions for both the DSN and DD file systems.

They apply equally as well to the Windows and UNIX operating systems, with appropriate changes for the file system syntactical differences.

### Transferring Files Examples - Windows and UNIX

- [Simple File Copy to the Manager - Windows and UNIX](#)
- [Simple File Copy to the Server - Windows and UNIX](#)
- [Copy a Set of Files - Windows and UNIX](#)

These examples illustrate two-party transfer sessions.

Each example illustrates a procedure that occurs under the default file system for that operating system.

See the list of z/OS and IBM i examples for file transfer examples that apply equally as well to the Windows operating systems.

### Transferring Files Examples - IBM i

- [Copy a File to an Existing IBM i File](#)
- [Copy an IBM i Data Physical File to a File](#)
- [Copy a Set of Files to an Existing Data Physical File](#)
- [Copy a File to a New IBM i Data Physical File](#)
- [Copy a File to a New IBM i Source Physical File](#)
- [Copy a Set of Files to a New Data Physical File on IBM i](#)
- [Copy Different Types of IBM i Files Using forfiles and \\$\\_file.type](#)
- [Invoke a Script from an IBM i Batch Job](#)



#### Note

These examples reference the IBM i commands by their untagged names. If you are using commands with tagged names to run [Universal Data Mover](#), substitute the tagged names for these untagged names. (For information on tagged names, see [UCHGRLS \(Change Release Tag\) Program](#).)

These examples illustrate two-party transfer sessions between IBM i and UNIX. Each example illustrate a file transfer for the LIB file system.

They apply equally as well to the Windows and UNIX operating systems, with appropriate changes for the file system syntactical differences.

The first example, [Copy a File to an Existing IBM i File](#), also includes a version specific to the HFS file system. For other examples similar to those used in the HFS file system, see [Transferring Files Examples - Windows and UNIX](#).

## Copy a File to an Existing z/OS Sequential Data Set

- Copy a File to an Existing z/OS Sequential Data Set
  - DD File System
  - DSN File System
  - Components

## Copy a File to an Existing z/OS Sequential Data Set

These examples copy, in text mode, one sequential file to another. This is the simplest form of data transfer.

### DD File System

```
//S1      EXEC UDMPRC
//APOUT  DD DSN=APP.DATA.DAILY,DISP=SHR
//UNVSCR DD *
1 set _echo=yes
2 set _halton=warn
3 open unix=sol9 user=top098 pwd=p100m
4 filesys local=dd
5 cd unix=/opt/app/data
6 mode type=text
7 copy unix=data10.txt local=APOUT
8 quit
/*
```

For this first z/OS example, the following is a line-by-line explanation:

1. Line 1 turns on command echo, which results in each command being written prior to processing.
2. Line 2 sets the error condition value on which script processing halts. Any error greater than or equal to warn halts script processing.
3. Line 3 opens a session between the local UDM Manager and a remote UDM server running on host **sol9**. The host **sol9** is given the logical name of **unix**. The **open** command also provides user credentials for the UDM server to verify and, if successfully verified, specifies the user ID with which the UDM server executes.
4. Line 4 changes the local file system from the default of DSN to DD. The file system type dictates the syntax and semantics of file specifications, such as in the **copy** command.
5. Line 5 changes the current directory of the UDM server **unix** running on host **sol9**.
6. Line 6 changes the transfer mode type from binary (the default) to text. Text mode transfers will translate between code pages (for example, ASCII and EBCDIC) and process the end-of-line characters.
7. Line 7 is the **copy** command that actually moves the data between systems. It copies file **data10.txt** on server **unix** to the local UDM Manager ddname APOUT. Recall that line 4 sets the local file system type to DD; hence, APOUT is referencing a ddname.
8. Line 8 executes the **quit** command, which closes all sessions and exits UDM with the highest exit code set.

### DSN File System

```
//S1      EXEC UDMPRC
//UNVSCR DD *
1 set _echo=yes
2 set _halton=warn
3 open unix=sol9 user=top098 pwd=p100m
4 cd unix=/opt/app/data
5 mode type=text
6 attrib local createop=replace
7 copy unix=data10.txt local='app.data.daily'
8 quit
/*
```

The DSN file system example is basically the same as the DD file system example, with these changes:

- Removal of the **filesys** command (line 4 in the DD file system example), since the default file system for the z/OS manager is DSN.
- Addition of line 6, which sets the local attribute **createop**. **createop** controls how a file is created. By default, its value is **new**, indicating that only new files are created and existing files are not written over (replaced). In this example, the value is being set to **replace**, which specifies that if the file exists, it should be replaced; otherwise, it is created.



## Components

Universal Data Mover Manager for z/OS

## Copy a File to a New z/OS Sequential Data Set

- [Copy a File to a New z/OS Sequential Data Set](#)
  - [DSN File System](#)
  - [Components](#)

### Copy a File to a New z/OS Sequential Data Set

This example copies, in text mode, a file from a remote UNIX system to a sequential data set on z/OS. The data set does not exist on z/OS; UDM is instructed to create it.

The data set is dynamically allocated based on the local UDM dynamic allocation attributes. UDM provides default attributes that can be changed to meet local requirements. The UDM defaults, as they are delivered, create a sequential, variable block record data set with a logical record length of 1024.

The sample below changes the record length to 256 in order to demonstrate how to set dynamic allocation attributes.

A DD file system sample is not provided, since creating a new data set with JCL is the same in UDM as it is in any batch application. There are no UDM specific requirements.

#### DSN File System

```
//S1      EXEC UDMPRC
//UNVSCR DD *
1 set _echo=yes _halton=warn
2 open unix=sol9 user=top098 pwd=p100m
3 cd unix=/opt/app/data
4 mode type=text
5 attrib local lrecl=256
6 copy data10.txt local='app.data.daily'
7 quit
/*
```



#### Note

All file names in the UNIX system must be within the eight-character range to be transferred successfully.

Almost all data set allocation attributes can be specified as UDM attributes, providing you with the ability to dynamically allocate any supported data set.

Care should be taken that conflicting allocation attributes are not specified. The results of the allocation should be checked to verify they meet your intentions. Although UDM checks attribute values, some values are provided by the system from sources that UDM cannot verify and can result in invalid or unintentional attribute combinations.

#### Components

[Universal Data Mover Manager for z/OS](#)

## Copy a z/OS Sequential Data Set to a File

- Copy a z/OS Sequential Data Set to a File
  - DD File System
  - DSN File System
  - Components

## Copy a z/OS Sequential Data Set to a File

These examples copy, in text mode, a sequential data set on z/OS to a remote UNIX system.



### Note

A text transfer, by default, does not trim spaces from the end of a record. If the data set being copied is a fixed record format, each record is padded with spaces so that the record length equals the logical record length of the data set. If you do not want the trailing spaces copied, they must be trimmed. Variable record formats do not normally have trailing spaces, so trimming normally is not required.

### DD File System

```
//S1      EXEC UDMPRC
//APOUT DD DSN=APP.DATA.DAILY,DISP=SHR
//UNVSCR DD *
1 set _echo=yes
2 set _halton=warn
3 open unix=sol9 user=top098 pwd=p100m
4 filesys local=dd
5 cd unix=/opt/app/data
6 mode type=text trim=yes
7 copy local=apout unix=data10.txt
8 quit
/*
```

### DSN File System

```
//S1      EXEC UDMPRC
//UNVSCR DD *
1 set _echo=yes
2 set _halton=warn
3 open unix=sol9 user=top098 pwd=p100m
4 cd unix=/opt/app/data
5 mode type=text trim=yes
6 copy local='app.data.daily' unix=data10.txt
7 quit
/*
```

### Components

Universal Data Mover Manager for z/OS

## Copy a Set of Files to an Existing z/OS Partitioned Data Set

- Copy a Set of Files to an Existing z/OS Partitioned Data Set
  - DD File System
  - DSN File System
  - Components

### Copy a Set of Files to an Existing z/OS Partitioned Data Set

These examples copy (in text mode, and using the \* wildcard) multiple files with one **copy** command to an already allocated partitioned data set (PDS) on a z/OS system.

The file names used to create the member names in the destination PDS are the source file names.

However, note that file names on UNIX and Windows file systems often have a file extension as part of their name. A file extension is a suffix separated from the file's base name with a period (for example, BASE.TXT). The period in the file extension is not a valid character in PDS member names, so UDM must be instructed to remove the file extensions before copying them into the PDS.

The **truncext** attribute is used to instruct UDM to remove file name extensions from the source file prior to using the name as the destination member name.

This example assumes that the remote UNIX directory **/opt/app/data** contains the following list of files:

- **data001.txt**
- **data002.txt**
- **data003.txt**
- **data004.pr**
- **data005.pr**

The result of the copy operation will create the following members in PDS **APP.DATA.PDS**:

- **DATA001**
- **DATA002**
- **DATA003**

### DD File System

```
//S1      EXEC UDMPRC
//APOUT  DD DSN=APP.DATA.PDS,DISP=SHR
//UNVSCR DD *
1 set _echo=yes _halton=warn
2 open unix=sol9 user=top098 pwd=p100m
3 filesys local=dd
4 cd unix=/opt/app/data
5 mode type=text
6 attrib local truncext=yes
7 copy unix=*.txt local=apout
8 quit
/*
```

### DSN File System

```
//S1      EXEC UDMPRC
//UNVSCR DD *
1 set _echo=yes _halton=warn
2 open unix=sol9 user=top098 pwd=p100m
3 cd unix=/opt/app/data
4 mode type=text
5 attrib local truncext=yes
6 copy unix=*.txt local='app.data.daily'
7 quit
/*
```

## Components

Universal Data Mover Manager for z/OS

## Copy a Set of Files to a New zOS Partitioned Data Set

- [Copy a Set of Files to a New zOS Partitioned Data Set](#)
  - [DSN File System](#)
  - [Components](#)

### Copy a Set of Files to a New zOS Partitioned Data Set

This example copies, in text mode, a set of files from a remote UNIX system to a partitioned data set on z/OS. The data set does not exist on z/OS; UDM is instructed to create it.

The data set is dynamically allocated based on the local UDM dynamic allocation attributes. UDM provides default attributes that can be changed to meet local requirements. The UDM defaults as they are delivered create a sequential, variable block record data set with a logical record length of 1024.

This example changes the data set organization from sequential (PS) to partitioned (PO) and adjusts the data set's space allocation to space units of cylinders, primary space to 1, secondary space to 2, and directory blocks to 10.

#### DSN File System

```
//S1 EXEC UDMPRC
//UNVSCR DD *
1 set _echo=yes _halton=warn
2 open unix=sol9 user=top098 pwd=p100m
3 cd unix=/opt/app/data
4 mode type=text
5 attrib local dsorg=po spaceunit=cyl primspace=1 secspace=2 +
6     dirblocks=10 truncext=yes
7 copy unix=*.txt local='app.data.pds'
8 quit
/*
```



#### Note

Line 5 is continued onto line 6 with the line continuation character (+).

#### Components

[Universal Data Mover Manager for z/OS](#)

## Simple File Copy to the Manager - Windows and UNIX

### Simple File Copy to the Manager

This example copies, in text mode, one file to another. This is the simplest form of data transfer.

```
1 set _echo=yes
2 set _halton=warn
3 open unix=sol9 user=top098 pwd=p100m
4 cd unix=/opt/app/data
5 mode type=text
6 copy unix=data10.txt
7 quit
```

For this UNIX and Windows example, the following is a line-by-line explanation:

1. Line 1 turns on command echo, which results in each command being written prior to processing.
2. Line 2 sets error condition value on which script process halts. Any error equal to or greater than 4 halts script processing. A value of 4 effectively means halt on any error or warning.
3. Line 3 opens a session between the local UDM Manager and a remote UDM Server running on host **sol9**. The host **sol9** is given the a logical name of **unix**. The **open** command also provides user credentials for the UDM Server to verify and, if success verified, specifies the user ID with which the UDM Server executes.
4. Line 4 changes the current directory of the UDM server **unix** running on host **sol9**.
5. Line 5 changes the transfer mode type from binary (the default) to text. Text mode transfers will translate between code pages and process the end-of-line characters.
6. Line 6 is the **copy** command that actually moves the data between systems. It copies file **data10.txt** on server **unix** to the local UDM Manager as **data10.txt**.
7. Line 7 executes the **quit** command, which closes all sessions and exits UDM with the highest exit code set.

### Components

[Universal Data Mover Manager for Windows](#)

[Universal Data Mover Manager for UNIX](#)

## Simple File Copy to the Server - Windows and UNIX

### Simple File Copy to the Server

This example copies, in text mode, a sequential data set on the UDM Manager machine to a remote UNIX system.

```
1 set _echo=yes
2 set _halton=warn
3 open unix=sol9 user=top098 pwd=p100m
4 cd unix=/opt/app/data
5 mode type=text
6 copy local=c:\data10.txt
7 quit
```

For this UNIX and Windows example, the following is a line-by-line explanation:

1. Line 1 turns on command echo, which results in each command being sent to stdout prior to processing.
2. Line 2 sets error condition value on which script process halts. Any error equal to or greater than 4 halts script processing. A value of 4 effectively means halt on any error or warning.
3. Line 3 opens a session between the local UDM Manager and a remote UDM server running on host **sol9**. The host **sol9** is given the a logical name of **unix**. The **open** command also provides user credentials for the UDM server to verify and, if success verified, specifies the user ID with which the UDM server executes.
4. Line 4 changes the current directory of the UDM server **unix** running on host **sol9**.
5. Line 5 changes the transfer mode type from binary (the default) to text. Text mode transfers will translate between code pages and process the end-of-line characters.
6. Line 6 is the **copy** command that actually moves the data between systems. It copies file **data10.txt** in the root directory on drive C of the Windows machine to the UNIX Server as **data10.txt**.
7. Line 7 executes the **quit** command, which closes all sessions and exits UDM with the highest exit code set.

### Components

[Universal Data Mover Manager for Windows](#)

[Universal Data Mover Manager for UNIX](#)



## Copy a Set of Files - Windows and UNIX

### Copy a Set of Files

This example copies (in text mode, and using the \* wildcard) multiple files with one copy.

It assumes that the remote UNIX directory **/opt/app/data** contains the following list of files:

- **data001.txt**
- **data002.txt**
- **data003.txt**
- **data004.pr**
- **data005.pr**

The following files will be created on the destination machine:

- **data001.txt**
- **data002.txt**
- **data003.txt**

The **truncext** attribute is used to instruct UDM to remove file name extensions from the source file prior to using the name as the destination member name.

```
1 set _echo=yes _halton=warn
2 open unix=sol9 user=top098 pwd=p100m
3 cd unix=/opt/app/data
4 mode type=text
5 attrib local truncext=yes
6 copy unix=*.txt
7 quit
```

### Components

[Universal Data Mover Manager for Windows](#)

[Universal Data Mover Manager for UNIX](#)

## Copy a File to an Existing IBM i File

- Copy a File to an Existing IBM i File
  - LIB File System
  - HFS File System
  - Components

### Copy a File to an Existing IBM i File

These examples copy, in text mode, one sequential file to another. This is the simplest form of data transfer.

#### LIB File System

```
1 set _echo=yes
2 set _halton=warn
3 open unix=sol19 user=top098 pwd=p100m
4 cd unix=/opt/app/data
5 mode type=text
6 copy unix=data10.txt local=MYLIB/APPDATA(DAILY)
7 quit
```

For this first IBM i example, the following is a line-by-line explanation:

1. Line 1 turns on command echo, which results in each command being sent to stdout prior to processing.
2. Line 2 sets the error condition value on which script processing halts. Any error greater than or equal to **warn** halts script processing.
3. Line 3 opens a session between the local UDM Manager and a remote UDM server running on host **sol19**. The host **sol19** is given the logical name of **unix**. The **open** command also provides user credentials for the UDM server to verify and, if successfully verified, specifies the user ID with which the UDM server executes.
4. Line 4 changes the current directory of the UDM server **unix** running on host **sol19**.
5. Line 5 changes the transfer mode type from binary (the default) to text. Text mode transfers will translate between code pages (for example, ASCII and EBCDIC) and process the end-of-line characters.
6. Line 6 is the **copy** command that actually moves the data between systems. It copies file **data10.txt** on server **unix** to the local UDM Manager library: MYLIB Data Physical File APPDATA member DAILY.
7. Line 7 executes the **quit** command, which closes all sessions and exits UDM with the highest exit code set.

#### HFS File System

```
1 set _echo=yes
2 set _halton=warn
3 open unix=sol19 user=top098 pwd=p100m
4 filesys local=hfs
5 cd unix=/opt/app/data
6 mode type=text
7 attrib local createop=replace
8 copy unix=data10.txt local=/opt/appdata
9 quit
```

This HFS file system example is basically the same as the LIB file system example, with these changes:

- Addition of line 4, which changes the local file system from the default of LIB to HFS. The file system type dictates the syntax and semantics of file specifications, such as in the **copy** command.
- Addition of line 7, which sets the local attribute **createop**. **createop** controls how a file is created. By default, its value is **new**, which indicates that only new files are created and existing files are not written over (replaced). In this case, its value is being set to **replace**, specifying that if the file exists, it should be replaced; otherwise, it is created.

#### Components

[Universal Data Mover Manager for IBM i](#)

## Copy an IBM i Data Physical File to a File

- [Copy an IBM i Data Physical File to a File](#)
  - [LIB File System](#)
  - [Components](#)

## Copy an IBM i Data Physical File to a File

This example copies, in text mode, a Data Physical File on IBM i to a remote UNIX system.



### Note

A text transfer, by default, does not trim spaces from the end of a record. If the data set being copied is a fixed record format, each record is padded with spaces so that the record length equals the logical record length of the data set. If you do not want the trailing spaces copied, they must be trimmed.

## LIB File System

```
1 set _echo=yes
2 set _halton=warn
3 open unix=sol9 user=top098 pwd=p100m
4 cd unix=/opt/app/data
5 mode type=text trim=yes
6 copy local=MYLIB/APPDATA(DAILY) unix=data10.txt
7 quit
```

## Components

[Universal Data Mover Manager for IBM i](#)

## Copy a Set of Files to an Existing Data Physical File

- Copy a Set of Files to an Existing Data Physical File
  - LIB File System
  - Components

### Copy a Set of Files to an Existing Data Physical File

This example copies (in text mode, and using the \* wildcard) multiple files with one `copy` command to an already allocated Data Physical File on an IBM i system.

The file names used to create the member names in the destination Data Physical File are the source file names. However, note that file names on UNIX and Windows file systems often have a file extension as part of their name. A file extension is a suffix separated from the file's base name with a period (for example, BASE.TXT). Member names are limited to 10 characters on the IBM i system, so UDM must be instructed to remove the file extensions before copying them into the file.

The **truncext** attribute is used to instruct UDM to remove file name extensions from the source file prior to using the name as the destination member name.

This example assumes that the remote UNIX directory **/opt/app/data** contains the following list of files:

- **data001.txt**
- **data002.txt**
- **data003.txt**
- **data004.pr**
- **data005.pr**

The result of the copy operation will create the following members in Data Physical File APPDATA:

- **DATA001**
- **DATA002**
- **DATA003**

### LIB File System

```
1 set _echo=yes _halton=warn
2 open unix=sol9 user=top098 pwd=p100m
3 cd unix=/opt/app/data
4 mode type=text
5 attrib local truncext=yes
6 copy unix=*.txt local=MYLIB/APPDATA
7 quit
```

### Components

Universal Data Mover Manager for IBM i

## Copy a File to a New IBM i Data Physical File

- [Copy a File to a New IBM i Data Physical File](#)
  - [LIB File System](#)
  - [Components](#)

### Copy a File to a New IBM i Data Physical File

This example copies, in text mode, a file from a remote UNIX system to a data physical file on IBM i. The Data Physical File does not exist on IBM i; UDM is instructed to create it.

The file type created defaults to a Data Physical File. The Data Physical File is allocated based on the local UDM allocation attributes. UDM provides default attributes that can be changed to meet local requirements. The UDM defaults, as delivered, create a Data Physical File with a logical record length of 92 and maximum members of 1.

This example changes the record length to 80, and the maximum members to unlimited (*nomax*), in order to demonstrate how to set allocation attributes.

### LIB File System

```
1 set _echo=yes _halton=warn
2 open unix=sol9 user=top098 pwd=p100m
3 cd unix=/opt/app/data
4 mode type=text
5 attrib local rcdlen=80 maxmbrs=nomax
6 copy unix=data10.txt local=MYLIB/APPDATA(DAILY)
7 quit
```

Almost all data set allocation attributes can be specified as UDM attributes giving you the ability to dynamically allocate any supported Data Physical File.

Care should be taken that conflicting allocation attributes are not specified. The results of the allocation should be checked to verify they meet your intentions. Although UDM checks attribute values, some values are provided by the system from sources that UDM cannot verify and can result in invalid or unintentional attribute combinations.

### Components

[Universal Data Mover Manager for IBM i](#)

## Copy a File to a New IBM i Source Physical File

- [Copy a File to a New IBM i Source Physical File](#)
  - [LIB File System](#)
  - [Components](#)

### Copy a File to a New IBM i Source Physical File

This example copies, in text mode, a file from a remote UNIX system to a Source Physical File on IBM i. The Source Physical File does not exist on IBM i; UDM is instructed to create it.

The Source Physical File is allocated based on the local UDM allocation attributes. UDM provides default attributes that can be changed to meet local requirements. The UDM defaults, as delivered, create a Data Physical File with a logical record length of 92 and maximum members of 1.

This example changes the file type to **src** in order to demonstrate how to set allocation attributes.

### LIB File System

```
1 set _echo=yes _halton=warn
2 open unix=sol9 user=top098 pwd=p100m
3 cd unix=/opt/app/data
4 mode type=text
5 attrib local filetype=src
6 copy unix=data10.txt local=MYLIB/APPDATA(DAILY)
7 quit
```

Almost all data set allocation attributes can be specified as UDM attributes giving you the ability to dynamically allocate any supported Data Physical File.

Care should be taken that conflicting allocation attributes are not specified. The results of the allocation should be checked to verify they meet your intentions. Although UDM checks attribute values, some values are provided by the system from sources that UDM cannot verify and may result in invalid or unintentional attribute combinations.

### Components

[Universal Data Mover Manager for IBM i](#)

## Copy a Set of Files to a New Data Physical File on IBM i

- Copy a Set of Files to a New Data Physical File on IBM i
  - LIB File System
  - Components

### Copy a Set of Files to a New Data Physical File on IBM i

This example copies (in text mode, and using the \* wildcard) a set of files from a remote UNIX system to a data physical file on IBM i. The data file does not exist on IBM i; UDM is instructed to create it.

The data set is allocated based on the local UDM allocation attributes. UDM provides default attributes that can be changed to meet local requirements. The UDM defaults, as delivered, create a data physical file with a logical record length of 92 and maximum members of 1.

This example changes the record length to 80 and the maximum members to unlimited (*nomax*).

#### LIB File System

```
1 set _echo=yes _halton=warn
2 open unix=sol9 user=top098 pwd=p100m
3 cd unix=/opt/app/data
4 mode type=text
5 attrib local maxmbrs=nomax rcdlen=80 truncext=yes
6 copy unix=*.txt local=MYLIB/APPDATA
7 quit
```

#### Components

Universal Data Mover Manager for IBM i

## Copy Different Types of IBM i Files Using forfiles and \$\_file.type

- Copy Different Types of IBM i Files using forfiles and \$\_file.type
  - LIB File System
  - Components

### Copy Different Types of IBM i Files using forfiles and \$\_file.type

Physical files are considered directories in UDM because they contain 1+ member. Save files are considered files because they do not contain any members. The `forfiles` statement and the variable `$_file.type` allow you to do a wildcard copy on both save and physical files in the LIB file system.

This example copies a mix of files (Save and Physical) from an IBM i system in a single operation, using the `forfiles` statement and the `$_file.type` variable attribute.

#### LIB File System

```
forfiles src=MYLIB/*
if $_file.type EQ directory
copy src=$_path\(*)
else
copy src=$_path
end
end
```

#### Components

Universal Data Mover Manager for IBM i



## Invoke a Script from an IBM i Batch Job

- Invoke a Script from an IBM i Batch Job
  - LIB file system
  - Components

### Invoke a Script from an IBM i Batch Job

To invoke a script included as an inline file in a database job, the call must specify **\*FIRST** as the database member name.

The following example illustrates both:

- Invocation of an inline script, **CALLME**, using the STRUDM command from a database job.
- Invocation of an inline script, **CALL1**, using the CALL command from a database job.

### LIB file system

```
//BCHJOB JOB(testcall) ENDSEV(10) OUTQ(mytest/UDMOUTQ) LOGCLPGM(*YES) LOG(2 20 \*SECLVL) MSGQ(*USRPRF)
//DATA FILE(CALL1) ENDCHAR(ENDDATAFILE)
print msg="I made it to call1 - an inline file"
ENDDATAFILE
//DATA FILE(CALLME) ENDCHAR(ENDDATAFILE)
OPEN S=AS400V5 USER=qatest PWD=****\* PORT=4311
CALL CALL1(*FIRST)
CLOSE
ENDDATAFILE
STRUDM SCRFILE(CALLME)
//ENDBCHJOB
```

### Components

Universal Data Mover Manager for IBM i

## Infitran - Remote Execution

## Remote Execution - Overview

### Remote Execution

These pages provide information on the Remote Execution features and functionality of the Infitran business solution.

Infitran provides access to Universal Command Remote Execution via the Universal Data Mover `exec` command. The `exec` command invokes the Universal Command Manager and provides parameters for passing a subset of the Universal Command Manager options.

The `exec` command executes system commands on remote machines if you have Universal Command (UCMD) Manager on the same system with the UDM Manager.

Remote Execution refers to the ability of initiating work from one system (the local system), which executes on another system (the remote system). The type of work executed on the remote system consists of most any type of work that the remote system supports, such as commands and scripts. The Universal Command component is used to execute work on the remote system.

### Remote Execution Components

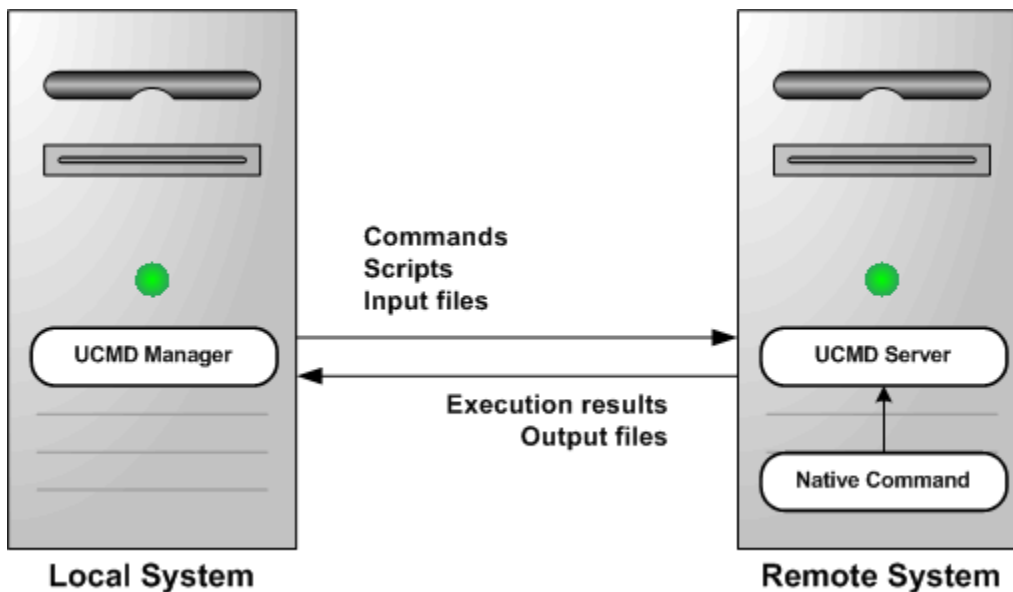
Infitran Remote Execution using Universal Command consists primarily of two Workload Automation 5 components:

1. Universal Command Manager runs on the local system. The Manager initiates the work on the remote system.
2. Universal Command Server runs on the remote systems. It executes work on behalf of a Universal Command Manager.

The Manager provides the information to the Server necessary for the Server to execute the work. This includes the command or script that defines the work, as well as the user identifier with which the work should execute. The Server authenticates the user identifier on the remote server. If the user identifier authenticates successfully, the Server executes the work with the provided user identifier.

Once the work is started, the Manager supplies input files to, and receives output files from, the remote command Server in real-time. All files with character data are translated to the appropriate code pages for the respective system. The transmitted data, optionally, can be compressed, encrypted, or authenticated.

The Manager runs as long as the remote work is running. When the remote work ends, the Manager ends. The exit code of the remote work is used as the exit code of the Manager. With standard out and standard error as well as the exit status of the remote work available from the manager, there is no need for access to or expertise on the remote operating system.



## Remote Execution - Primer

- [Overview](#)
- [Remote Execution Examples - exec Command Parameters](#)
- [Remote Execution Requirements](#)
  - [Executing the Examples](#)
- [Remote Execution Requirements for z/OS](#)
- [Remote Execution Requirements for Windows](#)
- [Remote Execution Requirements for UNIX](#)
- [Remote Execution Requirements for IBM i](#)

### Overview

This page discusses the basics of how to execute remote work using Infitran.



#### Note

Please read [Remote Execution - Overview](#) prior to reading this page, which builds upon the material presented in the Overview.

The primer discussions are from the perspective of the initiating system where the Universal Command (UCMD) Manager component is executed via the Universal Data Mover (UDM) `exec` command.

The primer examples assume that Infitran is installed with default configuration values to help keep the examples consistent and clear. UCMD components must be installed both on the local system from which the UCMD Manager is executed as well as the remote system where UCMD Server is executed.

The primer examples demonstrate how to execute a command on a remote system using the UDMD Manager component via the UDM Manager component using the UDM `exec` command. All examples use the same set of parameters.

### Remote Execution Examples - exec Command Parameters

The following table describes each of the parameters used in the following primer examples.

Parameter	Description
cmd	Command to be executed on the remote system.
user	Remote user ID with which to execute the command. The user ID must be a valid user ID on the remote system. The examples use a user ID value of <b>joe</b> . This will need to be changed to a valid user ID on the remote system on which Universal Command Server runs.
pwd	Password for the user ID on the remote system. The examples use an arbitrary value of <b>abcdefg</b> . This will need to be changed to the password for the USER_ID you use to execute the remote command.

### Remote Execution Requirements

This page illustrates the minimum set of parameters required to execute a remote process via the Universal Data Mover (UDM) `exec` command using UDM scripting language syntax.

The platform-independent nature of the UDM scripting language means that the format of `exec` is the same regardless of the UDM Manager's host platform. See the [Universal Data Mover 5.1.0 Reference Guide](#) for platform-specific information on executing UDM Manager, using UDM script files, and invoking `exec`.

`exec` instructs UDM to spawn a Universal Command (UCMD) Manager process. The [Universal Command 5.1.0 Reference Guide](#) contains platform-specific information for invoking UCMD Manager. A UCMD Server installed on the remote system receives the command specified by `exec`'s `cmd` parameter and executes it.

If security is enabled in the remote UCMD Server's configuration, `exec` must provide user account information. To establish a secure execution environment, the UCMD Server requires a user account ID, which `exec` specifies via the user parameter. The UCMD Server may also require a password (**pwd**) to authenticate the user account, depending on the remote operating system and Indesca configuration.

For information on securing access to Indesca components, see [Workload Automation 5 - Security](#).

## Executing the Examples

To execute the following examples in your environment, simply make these changes to the values specified in the command's parameters:

- Change the host name **dallas** or IP address 192.168.10.111 to a host name or IP address that exists in your environment.
- Change the cmd parameter to a valid system command or installed application on the remote system.
- Change the user ID **joe** to the name of a valid user account on the remote system.
- Change the password value **abcdefg** to the user account's password.

In each of these examples, the UCMD Manager establishes a network connection to the UCMD Server installed on the remote system (**dallas**). The UCMD Manager passes `exec` parameters to the UCMD Server over this connection. UCMD Server then executes the command as local user named **joe**.

The UCMD Manager and Server also establish network connections to forward the command's output (that is, everything it writes to standard output and standard error) to the UDM Manager. UDM Manager writes this output to its local standard output (stdout) and standard error (stderr) devices.

When the remote command completes, the UCMD Server retrieves the process' exit code and status and forwards them to the local UCMD Manager, which exits with that same value. The UDM Manager stores this exit code in its built-in `_execrc` variable.

## Remote Execution Requirements for z/OS

These examples illustrate how to execute a process on a remote z/OS system using the UDM `exec` command. Each example lists the contents of the `/u/joe` directory in the z/OS UNIX file system.

If no DNS entry is available for the remote host, use a statement like the one shown in [Remote z/OS Execution Using an IP Address](#); otherwise, use something similar to [Remote z/OS Execution Using a Host Name](#).

To execute a command on a remote system using an active UDM transfer session, follow the example shown in [Remote z/OS Execution Using a UDM Logical Session Name](#).

The `exec` command initiates UCMD Manager using the `UCMDPRC` JCL procedure installed in the `SUNVSAMP` library.

### Remote z/OS Execution Using an IP Address

```
exec 192.168.10.111 cmd="ls -al /u/joe" user=joe pwd=abcdefg
```

### Remote z/OS Execution Using a Host Name

```
exec dallas cmd=" ls -al /u/joe " user=joe pwd=abcdefg
```

### Remote z/OS Execution Using a UDM Logical Session Name

```
open rmtsys=dallas user=joe pwd= abcdefg
exec rmtsys cmd=" ls -al /u/joe "
```

## Remote Execution Requirements for Windows

These examples illustrate how to execute a process on a remote Windows system using the UDM `exec` command. Each example lists the contents of the `root` directory on the `c:` drive.

If no DNS entry is available for the remote host, use a statement similar to the one shown in [Remote Windows Execution Using an IP Address](#); otherwise, use something similar to [Remote Windows Execution Using a Host Name](#).

To execute a command on a remote system using an active UDM transfer session, follow the example shown in [Remote Windows Execution Using a UDM Logical Session Name](#).

### Remote Windows Execution Using an IP Address

```
exec 192.168.10.111 cmd="dir c:\" user=joe pwd=abcdefg
```

### Remote Windows Execution Using a Host Name

```
exec dallas cmd="dir c:\" user=joe pwd=abcdefg
```

### Remote Windows Execution Using a UDM Logical Session Name

```
open rmtsys=dallas user=joe pwd= abcdefg
exec rmtsys cmd="dir c:\"
```

## Remote Execution Requirements for UNIX

These examples illustrate how to execute a process on a remote UNIX system using the UDM `exec` command. Each example lists the contents of the home directory for the user account named `joe`.

If no DNS entry is available for the remote host, use a statement like the one shown in [Remote UNIX Execution Using an IP Address](#); otherwise, use something similar to [Remote UNIX Execution Using a Host Name](#).

To execute a command on a remote system using an active UDM transfer session, follow the example shown in [Remote UNIX Execution Using a UDM Logical Session Name](#).

### Remote UNIX Execution Using an IP Address

```
exec 192.168.10.111 cmd="ls -al /home/joe" user=joe pwd=abcdefg
```

### Remote UNIX Execution Using a Host Name

```
exec dallas cmd="ls -al /home/joe" user=joe pwd=abcdefg
```

### Remote UNIX Execution Using a UDM Logical Session Name

```
open rmtsys=dallas user=joe pwd= abcdefg
exec rmtsys cmd="ls -al /home/joe"
```

## Remote Execution Requirements for IBM i

These examples illustrate how to execute a process on a remote IBM i system using the UDM `exec` command. Each example lists the contents of the library `joelib`.

If no DNS entry is available for the remote host, use a statement like the one shown in [Remote IBM i Execution Using an IP Address](#); otherwise, use something similar to [Remote IBM i Execution Using a Host Name](#).

To execute a command on a remote system using an active UDM transfer session, follow the example shown in [Remote IBM i Execution Using a UDM Logical Session Name](#).

The `exec` command initiates UCMD Manager via runtime linkage on IBM i. Stonebranch only supports runtime linkage to the UCMD Manager using the `exec` command.

The operating system sends the output for the remote IBM i job to `QPRINT`. Use the Universal Submit Job utility (USBMJOB) to bring the output back to the local host via the UCMD Manager.

### Remote IBM i Execution Using an IP Address

```
exec 192.168.10.111 cmd="dsplib joelib" user=joe pwd=abcdefg
```

### Remote IBM i Execution Using a Host Name

```
exec dallas cmd=" dsplib joelib" user=joe pwd=abcdefg
```

### Remote IBM i Execution Using a UDM Logical Session Name

```
open rmtsys=dallas user=joe pwd= abcdefg
exec rmtsys cmd=" dsplib joelib"
```

## Remote Execution - Examples

### Remote Execution Examples - Windows

- Windows Directory Listing Using a Batch File - Default Directory
- Windows Directory Listing Using a Batch File - Returned File

### Remote Execution Examples - UNIX

- UNIX - Listing Using a Shell Script
- UNIX - Integrating UDM with FTP Using a Shell Script
- UNIX - Integrating UDM with FTP Using a Command Reference

### Remote Execution Examples - IBM i

- IBM i from Windows, UNIX, or IBM i - exec Command Return Codes

***In order to keep these examples as clear as possible, they do not check for error conditions. If any example is adopted for production use, it is recommended that you add appropriate error processing.***

## Windows Directory Listing Using a Batch File - Default Directory

- Windows Directory Listing Using a Batch File - Default Directory
  - UDM exec Command Parameters
  - Components

### Windows Directory Listing Using a Batch File - Default Directory

This example demonstrates using UCMD Manager via the UDM Manager `exec` command to provide a directory listing using a batch file.

The output from the batch file is redirected to the file `stdout.txt`. If this is not done, the output from the listing is output via UDM along with the Transaction Log. UDM creates the `stdout.txt` file in UDM's default directory, `Files\Universal\UCmdHome\joe`.



#### Note

The last directory in the path corresponds to the user ID under which the command is executed. No open state is used, and the remote host on the `exec` command is specified using the IP address.

```
set echo=yes
exec 192.168.20.47 cmd="C:\wrk\xmp\win\winxmp.bat > stdout.txt" user=joe pwd=abcdefg
quit
```

The `winxmp.bat` batch file simply does a `dir` command against the directory in which the batch file resides.

```
dir "C:\wrk\xmp\win"
```

Output sent to `stdout.txt`.

```
C:\Program Files\Universal\UCmdHome\manos>dir "C:\wrk\xmp\win"
Volume in drive C has no label.
Volume Serial Number is 3030-176B

Directory of C:\wrk\xmp\win

07/27/2011  03:27 PM    <DIR>          .
07/27/2011  03:27 PM    <DIR>          ..
07/27/2011  10:08 AM                20 winxmp.bat
07/27/2011  03:46 PM               106 winxmpbat.udm
                2 File(s)          126 bytes
                2 Dir(s)  13,453,979,648 bytes free
```

The transaction log is shown in this first example for those not used to seeing output from UDM.

```
2011.07.27 16.06.47.541 UNV2800I Universal Data Mover 5.1.0 Level 1 Release Build 105 started.
2011.07.27 16.06.47.556 Processing script: winxmpbat.udm
2011.07.27 16.06.47.556 exec 192.168.20.47 cmd="C:\wrk\xmp\win\winxmp.bat > stdout.txt" user=joe
pwd=\*
2011.07.27 16.06.48.431 quit
2011.07.27 16.06.48.447 Finished processing script: winxmpbat.udm
2011.07.27 16.06.49.447 UNV2801I Universal Data Mover 5.1.0 Level 1 Release Build 105 ended
successfully.
```

### UDM exec Command Parameters



The `exec` command parameters used in this example are:

Parameter	Description
cmd	Command to execute on the remote system using command type cmd (command).
user	Remote user ID with which to authenticate and execute the command on the remote system.
pwd	Password with which to authenticate the user ID on the remote system.

## Components

[Universal Data Mover Manager for Windows](#)

[Universal Command Server for Windows](#)

## Windows Directory Listing Using a Batch File - Returned File

- Windows Directory Listing Using a Batch File - Returned File
  - UDM exec Command Parameters
  - Components

### Windows Directory Listing Using a Batch File - Returned File

This example builds on the example illustrated in [Windows Directory Listing Using a Batch File - Default Directory](#).

Keep in mind that both the batch file and the file created by the redirected output reside on the remote system.

```

1 set echo=no
2 set outdir=C:\tmp\joe
3 open r=dallas user=joe pwd=abcdefg
4 exec r cmd="C:\wrk\xmp\win\winxmp.bat $(outdir)\stdout.txt" user=joe pwd=abcdefg
5 cd r=$(outdir)
6 cd local=C:\tmp\tmp
7 attrib local createop=replace
8 copy r=stdout.txt
9 exec local cmd="type C:\tmp\tmp\stdout.txt" user=joe pwd=abcdefg
10 quit

```

Due to the complexity of this example, each line (numbered for your convenience) is explained, below.

1. Echo is turned off to minimize the amount of information in the transaction log due to its size. You are encouraged to set up the example and work through the transaction log.
2. Set a variable, **outdir**, for later use. Instead of setting the variable inside of the UDM script, the variable and its associated value could have been provided externally via a script option.
3. Open the Infitran connection for a two-party transfer. The manager will act as the primary server and is known as **local**.
4. Execute the remote command passing the full path to the file for the redirected output. Note the use of the variable inside of the double quotations; this is a UDM feature.
5. Change the directory for the remote system to the directory in which **stdout.txt** resides.
6. Change the directory for the local system to the location in which you want **stdout.txt** to reside.
7. Set the attribute for the local system to allow replacement of the incoming file.
8. Perform the file copy.
9. Execute a command on the local system to display the contents of the received file. UCMD server runs on the local system just as it would on the remote system to execute the command.
10. Quit and exit the UCMD Manager.

The **winxmp.bat** batch file now echoes the received parameter. This puts output into the transaction log so that you can see what was passed to the remote system. The second line performs the **dir** command and redirects output to **stdout.txt**.

```

echo %1
dir "C:\wrk\xmp\win" > %1

```

Output sent to **stdout.txt**.

```

C:\Program Files\Universal\UCmdHome\joe>dir "C:\wrk\xmp\win"
Volume in drive C has no label.
Volume Serial Number is 3030-176B

Directory of C:\wrk\xmp\win

07/27/2011  03:27 PM    <DIR>          .
07/27/2011  03:27 PM    <DIR>          ..
07/27/2011  10:08 AM                20 winxmp.bat
07/27/2011  03:46 PM               106 winxmpbat.udm
                2 File(s)                126 bytes
                2 Dir(s)  13,453,979,648 bytes free

```

## UDM exec Command Parameters

The `exec` command parameters used in this example are:

Parameter	Description
<code>cmd</code>	Command to execute on the remote system using command type <code>cmd</code> (command).
<code>user</code>	Remote user ID with which to authenticate and execute the command on the remote system.
<code>pwd</code>	Password with which to authenticate the user ID on the remote system.

## Components

[Universal Data Mover Manager for Windows](#)

[Universal Command Server for Windows](#)

## UNIX - Listing Using a Shell Script

- UNIX Listing Using a Shell Script
  - UDM Script Explanation
  - UDM exec Command Parameters
  - Components

### UNIX Listing Using a Shell Script

In this example, the `exec` command runs on a UNIX system via UCMD Manager and executes the `sh` command to a remote UNIX system using UCMD Server. With a shell interpreter, such as Cygwin, installed under Windows, the same example would also apply to a Windows system. The example was tested using Linux as both the local and remote platforms.

Both the shell script and the file created by the shell script reside on the remote system. If you are walking through all the examples in order, notice that in this example the shell script redirects stdout to the `stdout.txt` file, whereas in the Windows example the command initiated by the remote UCMD server redirected stdout to the `stdout.txt` file.

Due to this difference, in this example `stdout.txt` is created in the current directory as set by the shell script and in the Windows example it is created in the UCMD server working directory.

```
1. set echo=yes
2. open r=houston user=joe pwd=abcdefg port=7887
3. exec r cmd="sh /home/joe/wrk/xmp/ls/ls.sh" user=joe pwd=abcdefg port=7887
4. quit
```

### UDM Script Explanation

1. Turns echo on to put the commands into the transaction log.
2. Open a connection to the remote UDM server using remote port 7887. This is the default port and can be changed by setting the port number in the Universal Broker configuration file on the remote system. When the port number is changed, Universal Broker on the remote system on which the configuration file change was made must be stopped and then started.
3. Execute the shell script on the remote system. The port must be specified on the command if it is set to a value other than the default value.
4. Quit command stops UDM script execution and the UDM script completes.

The shell script changes the current directory, generates the listing via the `ls` shell command, redirects the output of the `ls` command to the `stdout.txt` file and then uses the `cat` shell command to output the contents of `stdout.txt` to the stdout stream.

The stdout stream is returned by the UDM Server to the UDM Manager and is output to the transaction log.

```
cd /home/joe/wrk/xmp/ls
ls > stdout.txt
cat stdout.txt
```

Output sent to `stdout.txt`.

```
ls.sh
stdout.txt
```

Output sent to the UDM transaction log via stdout from the UDM Manager.

```

2011.07.28 10.13.06.845 UNV2800I Universal Data Mover 5.1.0 Level 0 Release Build 104 started.
2011.07.28 10.13.06.845 Processing script: ls.udm
2011.07.28 10.13.06.847 open r=houston user=joe pwd=* port=7887
2011.07.28 10.13.07.114 Data session established using cipher: NULL-MD5
2011.07.28 10.13.07.159 Two party session established with r (component 1278600806)
2011.07.28 10.13.07.161 Transfer mode settings:
2011.07.28 10.13.07.198     type=binary
2011.07.28 10.13.07.198     trim=no
2011.07.28 10.13.07.198 Session options:
2011.07.28 10.13.07.198     Keep Alive Interval:    120
2011.07.28 10.13.07.198     Network Fault Tolerant: yes
2011.07.28 10.13.07.198 exec r cmd="sh /home/joe/wrk/xmp/ls/ls.sh" user=joe pwd=* port=7887
ls.sh
stdout.txt

2011.07.28 10.13.08.072 quit
2011.07.28 10.13.08.074 Session closed
2011.07.28 10.13.08.074 Finished processing script: ls.udm
2011.07.28 10.13.10.074 UNV2801I Universal Data Mover 5.1.0 Level 0 Release Build 104 ended
successfully.

```

## UDM exec Command Parameters

The `exec` command parameters used in this example are:

Parameter	Description
cmd	Command to execute on the remote system using command type cmd (command).
user	Remote user ID with which to authenticate and execute the command on the remote system.
pwd	Password with which to authenticate the user ID on the remote system.
port	Port that the Universal Broker is listening on for the remote machine. The port is inherited from the UDM Manager's configuration file unless explicitly overridden in the call to the <code>exec</code> command

## Components

[Universal Data Mover Manager for UNIX](#)

[Universal Command Server for UNIX](#)

## UNIX - Integrating UDM with FTP Using a Shell Script

- UNIX - Integrating UDM with FTP Using a Shell Script
  - UDM `exec` Command Parameters
  - Components

### UNIX - Integrating UDM with FTP Using a Shell Script

Remote process may require coordination with UDM. The `exec` command provides a method for this coordination.

In this example, a file is transferred into a secure area behind a firewall and then is forwarded to a second system using FTP. In actual practice, the same file could be forwarded to multiple systems using FTP, and then the `exec` command used to send notices to those same systems.

For simplicity, the file is "pulled" to the local system using UDM and then "pushed" to the remote system inside of the firewall using FTP. Infitran's three-party transfer capability allows transferring a file from one remote system to another and initiating processes on either of those remote systems, the local system, or any other system running a UCMD Server.

The example was tested using a Windows system as the remote system from which the file is initially pulled. The example would work without change if the remote system were a UNIX system. The local test system on which the UDM Manager runs is Linux and the test system to which the file is sent using FTP is also Linux.

```

1. set echo=yes
2. open rmt=192.168.20.47 user=joe pwd=abcdefg port=7887
3. mode type=text
4. attrib local createop=replace
5. cd rmt=C:\tmp\tmp
6. cd local=/home/joe/wrk/xmp/dmzFtp
7. copy rmt=file.txt.org local=file.txt
8. exec local cmd="sh /home/joe/wrk/xmp/dmzFtp/ftp.sh" user=joe pwd=abcdefg port=7887
9. exec dev-linux24 cmd="ls /home/joe/tmp" user=joe pwd=abcdefg port=7887
10. quit

```

The shell script sets up and executes FTP commands.

```
ftp -ipnv houston <
```

### UDM exec Command Parameters

The `exec` command parameters used in this example are:

Parameter	Description
cmd	Command to execute on the remote system using command type cmd (command).
user	Remote user ID with which to authenticate and execute the command on the remote system.
pwd	Password with which to authenticate the user ID on the remote system.
port	Port that the Universal Broker is listening on for the remote machine. The port is inherited from the UDM Manager's configuration file unless explicitly overridden in the call to the <code>exec</code> command

### Components

Universal Data Mover Manager for UNIX

Universal Command Server for UNIX

## UNIX - Integrating UDM with FTP Using a Command Reference

- UNIX - Integrating UDM with FTP Using a Command Reference
  - UDM Script Explanation
  - UDM exec Command Parameters

### UNIX - Integrating UDM with FTP Using a Command Reference

This example demonstrates the use of Command Reference files. Command References provides a very secure environment in which to store and from which to execute commands and scripts for use with UCMD Manager.



#### Note

This example is based on the example in [UNIX - Integrating UDM with FTP Using a Shell Script](#). Understanding that example is a prerequisite to using this one. Also, the test environment in the previous example is the same as in this example.

If you are not familiar with Command References, please read [Command References](#).

#### UDM Script Explanation

Other than Line 8, this UDM script is identical to the previous example. The `exec` command in line 8 uses the UCMD server running on the local system to execute the shell script contained in the Command Reference file `ftp.cref`. One option, the remote system name, is passed to the script via the Command Reference.

Command Reference files must reside in the directory specified by the `CMD_REFERENCE_DIRECTORY` UCMD Server configuration option. On UNIX systems this directory defaults to `/var/opt/universal/cmdref`.

```
1. set echo=yes
2. open rmt=192.168.20.47 user=joe pwd=abcdefg port=7887
3. mode type=text
4. attrib local createop=replace
5. cd rmt=C:\tmp\tmp
6. cd local=/home/joe/wrk/xmp/dmzFtp
7. copy rmt=file.txt.org local=file.txt
8. exec local cmdref="ftp.cref houston" user=joe pwd=abcdefg port=7887
9. exec houston cmd="ls /home/joe/tmp" user=joe pwd=abcdefg port=7887
10. quit
```

The `ftp.cref` Command Reference file contains the shell script used to FTP the file to the remote system behind the firewall. The `allow_options` option is changed to **yes** to allow the server address to be passed to the script. By default, no options are passed.

The format option is changed from `cmd` to `script`; otherwise, the script will not be generated.

```
# Command reference to read a file.
#
-format script
-type shell
-allow_options yes

ftp -ipnv $1 <
```

#### UDM exec Command Parameters

The `exec` command parameters used in this example are:

Parameter	Description



cmdref	Command Reference file name and, optionally, options to be passed to the command or script.
user	Remote user ID with which to authenticate and execute the command on the remote system.
pwd	Password with which to authenticate the user ID on the remote system.
port	Port that the Universal Broker is listening on for the remote machine. The port is inherited from the UDM Manager's configuration file unless explicitly overridden in the call to the <code>exec</code> command

## IBM i from Windows, UNIX, or IBM i - exec Command Return Codes

- IBM i from Windows, UNIX, or IBM i - exec Command Return Codes
  - UDM Script Explanation
  - Operating System-Specific Information
  - UDM exec Command Parameters
  - Components

### IBM i from Windows, UNIX, or IBM i - exec Command Return Codes

This example demonstrates using the `_execrc` built-in variable.

For IBM i, the UCMD Server checks the error severity for each CL command issued. If the severity of the error exceeds the value set via the UCMD Server `END_SEVERITY` option, the value is returned via `_execrc`. A UCMD Server error may also result in `_execrc` being set. If no error occurs, `_execrc` is zero.

Generally, UCMD Server return codes for IBM i are 200 or greater. Therefore, return codes associated with `END_SEVERITY` and with the UCMD Server do not conflict.

The `svropt` parameter passes options to the UCMD Server. These options override both the defaults and the options contained in the UCMD Server configuration file. The `-joblog never` value prevents the job log from being returned to the transaction log via stdout. (Do not include `svropt` if you want the job log.) The spaces before and after the double quotation marks are significant. `END_SEVERITY` can also be overridden.

The `exec` commands are both broken into two lines. The `-` and `+` characters are line continuation characters. Using `-` trims all leading blanks from the beginning of the next line; using `+` retains the blanks. In the example script, only one blank remains to separate the text on the two lines after they are concatenated.

This UDM example script was tested on three different platforms: Linux, Windows XP, and IBM i.

```

1. set echo=yes
2. exec atlanta cmd="SAVLIB LIB(NONAME) DEV(*SAVF) SAVF(QGPL/ABC)" user=joe -
   pwd=abcdefg port=27887 svropt=" \-joblog never "
3. echo "rc = " $_execrc
4. if $_execrc GE 30
5.   exec atlanta cmd="SNDMSG MSG('The command, SAVLIB LIB(NONAME) DEV(*SAVF) -
   SAVF(QGPL/ABC), failed') TOUSR(*SYSOPR)" user=joe pwd=abcdefg port=27887 svropt=" \-joblog
   never "
6. end
7. quit

```

### UDM Script Explanation

The script issues an IBM i command that fails and, based on the failure, issues an IBM i command to notify the system operator.

1. Turns echo on.
2. Issues a SAVLIB command to system atlanta which fails with end severity 40.
3. Echoes the value returned to the UDM Manager from the system via the UCMD Server.
4. Checks for the error.
5. Issues the SNDMSG command to notify the system operator.
6. Closes the if statement.
7. Cleans up and exits the UDM script.

### Operating System-Specific Information

Although the same script works equally well on Windows, UNIX, and IBM i, the syntax for submitting the script differs.

<b>Windows and UNIX</b>	The syntax is <b>udm -s script-path</b> . To run the example, change the current directory to the location of the script and issue <b>udm -s xmp0.udm</b> , where <b>xmp0.udm</b> is the name of the file containing the script.
<b>IBM i</b>	The syntax is <b>STRUDM qualified-file-name file-member-name</b> . To run the example, enter <b>STRUDM joe/qscsrc xmp0_udm</b> . The file and member names are positional parameters. <b>STRUDM SCRFILE(JOE/QSCRSRC) SCRMBR(XMP0_UDM)</b> is also valid.

## UDM exec Command Parameters

The `exec` command parameters used in this example are:

Parameter	Description
<code>cmd</code>	Command Reference file name and, optionally, options to be passed to the command or script.
<code>user</code>	Remote user ID with which to authenticate and execute the command on the remote system.
<code>pwd</code>	Password with which to authenticate the user ID on the remote system.
<code>port</code>	Port that the Universal Broker is listening on for the remote machine. The port is inherited from the UDM Manager's configuration file unless explicitly overridden in the call to the <code>exec</code> command
<code>svropt</code>	Server option to pass to the UCMD server.

## Components

[Universal Data Mover Manager for IBM i](#)

[Universal Data Mover Manager for Windows](#)

[Universal Data Mover Manager for UNIX](#)

[Universal Command Manager for IBM i](#)

## **Infitran - Remote Execution for SAP Systems**

## Remote Execution for SAP Systems - Overview

### Remote Execution for SAP Systems

These pages provides information on the Remote Execution for SAP feature and functionality of the Infitran business solution.

With Infitran, Remote Execution for SAP systems is performed indirectly. Infitran provides the ability to raise events within the remote SAP system. These events can be used by the SAP scheduling system to trigger job runs. This allows the automated coordination of work on the SAP system from within an Infitran process.

### Remote Execution for SAP Examples

The remote execution for SAP [examples](#) illustrated in these pages are specific to the operating systems supported by Workload Automation 5 for the Remote Execution for SAP feature of Infitran. The examples demonstrate the use of Universal Connector *for Use with SAP® ERP* to define SAP jobs.

Links to detailed technical information on appropriate Iniftran components are provided for each example.

## Remote Execution for SAP Systems - Examples

### Remote Execution for SAP Systems Examples

- [Raising an SAP Event for z/OS Example](#)
- [Raising an SAP Event for UNIX Example](#)

*These examples illustrate the Remote Execution of SAP feature of Infitran using the Universal Data Mover `execsap` command.*

## Raising an SAP Event for zOS Example

- Raising an SAP Event for z/OS Example
  - Raising an SAP Event for z/OS JCL
  - Components

### Raising an SAP Event for z/OS Example

The following example demonstrates raising events in a remote SAP system using the Universal Data Mover [execsap](#) command.

In this example, we assume the following scenario:

The job scheduler on SAP system CF5 has been set up with three jobs that are triggered by SAP event UDM\_TRANSFER\_COMPLETE. Additionally, each job is looking for a different event parameter (**cars.dat**, **trucks.dat**, and **boats.dat**) corresponding with the Input file it is intended to process.

Infitran is being run on a z/OS system to transfer three data files (**cars.dat**, **trucks.dat**, and **boats.dat**) from remote system **sol9** to remote system **SAP001**. The data files are to be used by the SAP system for Batch Input Processing. Therefore, after each file transfer, the [execsap](#) command is issued to raise an appropriate event in the SAP system. These events are picked up by the SAP job scheduler which, in turn, kicks off the jobs that were scheduled for those events.

### Raising an SAP Event for z/OS JCL

```

//UDMEXSAP JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
/*****\
/* Description
/* -----
/* This sample opens a three-party transfer session between hosts
/* sol9 and SAP001. Three files are transferred from sol9 to
/* SAP001. After each file is transferred, execsap is called to
/* raise an SAP event in the specified SAP system.
/*
/* Presumably, there are jobs in the SAP scheduling system that
/* are waiting to be triggered by the events fired from this job.
/*
//          JCLLIB ORDER=#SHLQ.UNV.SUNVSAMP
/*
//STEP1      EXEC UDMPRC
//UNVSCR     DD      *
#
# Transfer vehicle data to SAP server for batch input processing.
#
open src=sol9 dest=SAP001 xfile=xuser1

attrib dest createop=replace
cd src=/opt/app/data
cd dest=/input

*****\
/* Copy the car data to SAP system for batch input processing.
*****\
copy src=cars.dat dest=cars.dat

# Raise SAP event to trigger processing job.
execsap CF5 client=800 xfile=xsapuser1 type=event -
        eventid=UDM_TRANSFER_COMPLETE parm="cars.dat"

*****\
/* Copy the truck data to SAP system for batch input processing.
*****\
copy src=trucks.dat dest=trucks.dat

# Raise SAP event to trigger processing job.
execsap CF5 client=800 xfile=xsapuser1 type=event -
        eventid=UDM_TRANSFER_COMPLETE parm="trucks.dat"

*****\
/* Copy the boat data to SAP system for batch input processing.
*****\
copy src=boats.dat dest=boats.dat

# Raise SAP event to trigger processing job.
execsap CF5 client=800 xfile=xsapuser1 type=event -
        eventid=UDM_TRANSFER_COMPLETE parm="boats.dat"

close
/*

```

## Components

Universal Data Mover Manager for z/OS

Universal Data Mover Server for UNIX

Universal Connector for z/OS



## Raising an SAP Event for UNIX Example

- Raising an SAP Event for UNIX Example
  - Raising an SAP Event for UNIX - UDM Script File: BIVehicle001
  - Components

### Raising an SAP Event for UNIX Example

The following example demonstrates raising events in a remote SAP system using the Universal Data Mover `execsap` command.

In this example, we assume the following scenario:

The job scheduler on SAP system CF5 has been set up with three jobs that are triggered by SAP event UDM\_TRANSFER\_COMPLETE. Additionally, each job is looking for a different event parameter (**cars.dat**, **trucks.dat**, and **boats.dat**) corresponding with the Input file it is intended to process.

Infitran is being run on a UNIX system to transfer three data files (**cars.dat**, **trucks.dat**, and **boats.dat**) from remote system **sol9** to remote system **SAP001**. The data files are to be used by the SAP system for Batch Input Processing. Therefore, after each file transfer, the `execsap` command is issued to raise an appropriate event in the SAP system. These events are picked up by the SAP job scheduler which, in turn, kicks off the jobs that were scheduled for those events.

### Raising an SAP Event for UNIX - UDM Script File: BIVehicle001

```

*****\*
# Description
# -----
# This sample opens a three-party transfer session between hosts
# sol9 and SAP001. Three files are transferred from sol9 to
# SAP001. After each file is transferred, execsap is called to
# raise an SAP event in the specified SAP system.
#
# Presumably, there are jobs in the SAP scheduling system that
# are waiting to be triggered by the events fired from this job.
#

open src=sol9 dest=SAP001 xfile=xuser1
attrib dest createop=replace
cd src=/opt/app/data
cd dest=/input

*****\*
#* Copy the car data to SAP system for batch input processing.
*****\*
copy src=cars.dat dest=cars.dat

# Raise SAP event to inform the system that the input file is ready.
execsap CF5 client=800 xfile=xsapuser1 type=event -
      eventid=UDM_TRANSFER_COMPLETE parm="cars.dat"

*****\*
#* Copy the truck data to SAP system for batch input processing.
*****\*
copy src=trucks.dat dest=trucks.dat

# Raise SAP event to inform the system that the input file is ready.
execsap CF5 client=800 xfile=xsapuser1 type=event -
      eventid=UDM_TRANSFER_COMPLETE parm="trucks.dat"

*****\*
#* Copy the boat data to SAP system for batch input processing.
*****\*
copy src=boats.dat dest=boats.dat

# Raise SAP event to inform the system that the input file is ready.
execsap CF5 client=800 xfile=xsapuser1 type=event -
      eventid=UDM_TRANSFER_COMPLETE parm="boats.dat"

close

```

## **Components**

Universal Data Mover Manager for UNIX

Universal Data Mover Server for zOS

Universal Connector for zOS

## **Infitran - Web Services Execution**

## Web Services Execution - Overview

### Infitran Web Services Execution

The inbound implementation of Infitran's web services execution – Universal Event Monitor for SOA – provides the ability to create file-based events from inbound Internet and message-based messages, and write the events to file.

This allows for the integration of Internet and message-based messages, and write the events to file. As such it integrates Internet and message-based applications with systems management functions such as:

- Alerting and notification
- Incident and problem management
- Job scheduling
- Data movement

Universal Event Monitor (UEM) monitors 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 is monitoring.

## Web Services Execution - Examples

### Web Services Execution Examples

- [Inbound JMS Implementation](#)
- [Inbound SOAP Implementation](#)

## Inbound JMS Implementation

- Inbound JMS Implementation
  - ActiveMQ Topic
  - Websphere Queue
  - MQ Series Queue
  - Triggering an Event
  - Components

### Inbound JMS Implementation

Inbound implementations take the form of modifying the **UAC.xml** file with a definition. The container will read this file to construct the connection to the target defined by the value of the Property **java.naming.provider**.

The following figure illustrates an example of this construction.

```
<sb:Property>
    <sb:Name>java.naming.provider.url</sb:Name>
    <sb:Value>tcp://soatest2:61616</sb:Value>
</sb:Property>
```

In the following examples:

- Messages consumed from the topic or queue are written to the file system defined by the **<sb:Directory>** tag.
- **<sb:Filename>** tag denotes the filename that is be written to the filesystem.
- **%Seq%** defines an increment so that subsequent messages consumed from the topic do not collide with already existing filenames.

### ActiveMQ Topic

The following figure illustrates an attachment to an Apache ActiveMQ dynamic topic.

```
<sb:JMSConnection>
  <sb:Name>JMS ActiveMQ Topic Listener - soatest2</sb:Name>
  <sb:InitialContextProperties>
    <sb:Property>
      <sb:Name>java.naming.factory.initial</sb:Name>
      <sb:Value>org.apache.activemq.jndi.ActiveMQInitialContextFactory</sb:Value>
    </sb:Property>
    <sb:Property>
      <sb:Name>java.naming.provider.url</sb:Name>
      <sb:Value>tcp://soatest2:61616</sb:Value>
    </sb:Property>
  </sb:InitialContextProperties>
  <sb:ConnectionFactory>ConnectionFactory</sb:ConnectionFactory>
  <sb:Listeners>
    <sb:JMSListener>
      <sb:Destination>dynamicTopics/UemsoaStartTopicA</sb:Destination>
      <sb:Actions>
        <sb:JMSFileWriter>
          <sb:Directory>filesystem</sb:Directory>
          <sb:FilenamePattern>ActiveMQ_Topic_%Seq%.txt</sb:FilenamePattern>
          <sb:StartSequenceNumber>1</sb:StartSequenceNumber>
          <sb:WriteProperties>>false</sb:WriteProperties>
        </sb:JMSFileWriter>
      </sb:Actions>
    </sb:JMSListener>
  </sb:Listeners>
</sb:JMSConnection>
</pre>
```

### Websphere Queue

The following figure illustrates an attachment to an IBM Websphere queue.

```

<sb:JMSConnection>
  <sb:Name>JMS WebSphere Queue Listener - soatest2</sb:Name>
  <sb:InitialContextProperties>
    <sb:Property>
      <sb:Name>java.naming.factory.initial</sb:Name>
      <sb:Value>com.ibm.websphere.naming.WsnInitialContextFactory</sb:Value>
    </sb:Property>
    <sb:Property>
      <sb:Name>java.naming.provider.url</sb:Name>
      <sb:Value>iio://soatest2:2809</sb:Value>
    </sb:Property>
    <sb:Property>
      <sb:Name>com.ibm.CORBA.ORBInit</sb:Name>
      <sb:Value>com.ibm.ws.sib.client.ORB</sb:Value>
    </sb:Property>
  </sb:InitialContextProperties>
  <sb:ConnectionFactory>jms/SBSCONNECTIONFACTORY</sb:ConnectionFactory>
  <sb:Listeners>
    <sb:JMSListener>
      <sb:Destination>jms/Soatest2TestQueue3</sb:Destination>
      <sb:Actions>
        <sb:JMSFileWriter>
          <sb:Directory>filesystem</sb:Directory>
          <sb:FilenamePattern>WebSphere_Queue_%Seq%.txt</sb:FilenamePattern>
          <sb:StartSequenceNumber>1</sb:StartSequenceNumber>
          <sb:WriteProperties>>false</sb:WriteProperties>
        </sb:JMSFileWriter>
      </sb:Actions>
    </sb:JMSListener>
  </sb:Listeners>
</sb:JMSConnection>

```

## MQ Series Queue

The following figure illustrates an attachment to an IBM MQ Series Queue.

```

<sb:MQConnection>
  <sb:Name>MQ Series Listener - soatest2</sb:Name>
  <sb:Host>soatest2</sb:Host>
  <sb:QueueManagerName>MyQueueManager</sb:QueueManagerName>
  <sb:Channel>UpsQaChannel</sb:Channel>
  <sb:Port>1414</sb:Port>
  <sb:Listeners>
    <sb:MQListener>
      <sb:QueueName>UpsQaQueue</sb:QueueName>
      <sb:Actions>
        <sb:MQFileWriter>
          <sb:Directory>filesystem</sb:Directory>
          <sb:FilenamePattern>MQSeries_Queue_%Seq%.txt</sb:FilenamePattern>
          <sb:StartSequenceNumber>0</sb:StartSequenceNumber>
          <sb:WriteProperties>>false</sb:WriteProperties>
        </sb:MQFileWriter>
      </sb:Actions>
    </sb:MQListener>
  </sb:Listeners>
</sb:MQConnection>

```

## Triggering an Event

Once a file has been written to the file system, UEM could be used to trigger an event, as shown in the following figure.

This event, which would be loaded by UEMLoad, looks for files with an extension of **txt**. When it sees a file with that extension, UEM renames the file to the original name with an **xml** extension. It then executes the handler, which runs a system command to move the file.

The UDM script looks for all files that begin with a 2 and end with .xml on the local server. These file are then transferred to the destination server, overwriting any existing files on the destination server, and the session is closed.

```

begin_event
  event_id "JMS_MESSAGE_TRIGGER"
  event_type FILE
  comp_name uems
  state enable
  tracking_int 10
  triggered_id "JMS_MESSAGE_HANDLER"
  filespec "filesystem/*.txt"
  min_file_size 0
  rename_file yes
  rename_filespec "filesystem/${origname}.xml"
end_event

begin_handler
  handler_id "JMS_MESSAGE_HANDLER"
  handler_type CMD
  maxrc 0
  userid username
  pwd user_password
  cmd "udm -s udm.script"
end_handler

```

### Event Options

The Event options used in this example are:

Option	Description
event_id	Identifier that uniquely identifies an event definition record.
event_type	Type of system event represented by the event definition record.
comp_name	Event-driven UEM Server responsible for monitoring the event.
state	Event definitions that should be processed or ignored by UEM.
tracking_int	Event definitions that should be processed or ignored by UEM.
triggered_id	ID of an event handler record that UEM will execute when an event occurrence is triggered.
filespec	Name of a file to monitor.
min_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_filespec	Specifies how a file should be renamed when an event occurrence is triggered.
handler_id	Identifier that uniquely identifies an event handler record.



<code>handler_type</code>	Type of process executed for the event handler.
<code>maxrc</code>	Highest value with which a handler can exit to still be considered as having executed successfully.
<code>userid</code>	ID of a user account in whose security context the handler process will be executed.
<code>pwd</code>	Password for the user account specified by <code>userid</code> .
<code>cmd</code>	Command to execute on behalf of the event handler.

### **Contents of File `udm.script`**

```
open dest_server=192.168.1.1 user=gatest pwd=gatest
attrib dest_server createop=replace
forfiles local=2*.xml
  copy local=${_file}
end
close
```

### **Components**

Universal Event Monitor

UEMLoad

Universal Event Monitor for SOA

## Inbound SOAP Implementation

- Inbound SOAP Implementation
  - Inbound SOAP Request UAC.xml
  - Inbound SOAP Request - Message Payload Written to **process\_%Seq%.xml** File
  - Inbound SOAP Request - Universal Event Monitor Event Definition
    - Event Definition Options
  - Loading the Event Definition
    - Command Line Options
  - Changing the Event Definition
    - Command Line Options
  - Inbound SOAP Request - Universal Event Monitor Handler Definition
  - Outbound SOAP Request - abc.rexx
  - Outbound SOAP Request - Event and Handler to purge abc.log
    - Event Options
  - Components

## Inbound SOAP Implementation

Inbound SOAP requests are handled via Universal Event Monitor for SOA.

When Universal Event Monitor for SOA detects an inbound SOAP message, it writes the message payload to a file. Universal Event Monitor detects the file and initiates an action.

The SOAP message payload is parsed to extract information that is used to build a z/OS console message. Universal Command delivers the message from the Linux server to the z/OS mainframe.

Universal Event Monitor for SOA is configured via the `/etc/universal/UAC.xml` file.

### Inbound SOAP Request UAC.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<sb:UAC xmlns:sb="http://com.stonebranch/UAC/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://com.stonebranch/UAC/ UAC.xsd" /
  <!-- $Id$ -->
  <sb:SOAPConnection>
    <sb:URI>/axis2/services/UACInbound</sb:URI>
    <sb:Listeners>
      <sb:SOAPListener>
        <sb:Operation>process</sb:Operation>
        <sb:Actions>
          <sb:SOAPFileWriter>
            <sb:Directory>/export/home/control/indesca/soap_listener/</sb:Directory>
            <sb:FilenamePattern>process_%Seq%.xml</sb:FilenamePattern>
            <sb:StartSequenceNumber>1</sb:StartSequenceNumber>
            <sb:WriteEnvelope>true</sb:WriteEnvelope>
          </sb:SOAPFileWriter>
        </sb:Actions>
      </sb:SOAPListener>
    </sb:Listeners>
  </sb:SOAPConnection>
</sb:UAC>
```

If required, additional SOAP connections can be defined to the **UAC.xml**.

Universal Event Monitor for SOA writes the payload of the inbound SOAP message to the following directory / file mask:

`/export/home/control/indesca/soap_listener/process_%Seq%.xml`

The variable **%Seq%** is resolved to a sequence number generated by Universal Event Monitor. The sequence number is incremented by one for each file created and is reset to 1 each time Universal Event Monitor for SOA is started.

### Inbound SOAP Request – Message Payload Written to process\_%Seq%.xml File

The following shows an example of the inbound message payload written to the **process\_%Seq%.xml** file.

```
<?xml version='1.0' encoding='utf-8'?><soapenv:Envelope xmlns:soapenv=
"http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"><soapenv:Body><NS1:process
xmlns:NS1="http://inbound.uac.stonebranch.com">
<NS1:identitySourceApplicationId>RBS</NS1:identitySourceApplicationId><NS1:identitySourceUserId />
<NS1:identitySourcePassword /><NS1:identitySourceToken />
<NS1:activityRequestId>AUT4210021109265970293000</NS1:activityRequestId>
<NS1:activityStatus>PROCESS CLOSE ACCOUNTING YYYY MM</NS1:activityStatus>
<NS1:activityState>ACCOUNTING MONTH CLOSING INPROGRESS</NS1:activityState>
<NS1:activityStateReason>INFO</NS1:activityStateReason>
<NS1:activityAction>ODPT0001</NS1:activityAction>
<NS1:activityStartDate>2010\02\24</NS1:activityStartDate>
<NS1:activityStartTime>08:35:42.397382</NS1:activityStartTime></NS1:process></soapenv:Body></soapenv:Envelope>
```

The following fields in the `process_%Seq%.xml` file are used to create the z/OS console message:

- `<NS1:identitySourceApplicationId>RBS</NS1:identitySourceApplicationId>`
- `<NS1:activityRequestId>AUT4210021109265970293000</NS1:activityRequestId>`
- `<NS1:activityAction>ODPT0001</NS1:activityAction>`

### Inbound SOAP Request – Universal Event Monitor Event Definition

The following figure illustrates the event definition that Universal Event Monitor uses to detect the file created by Universal Event Monitor for SOA.

```
BEGIN_EVENT
EVENT_ID           "ABC SOA EVENT"
EVENT_TYPE        FILE
COMP_NAME         UEMS
STATE             ENABLE
TRACKING_INT      10
TRIGGERED_ID     "ABC SOA HANDLER"

FILESPEC          "/export/home/ control/indesca/soap_listener/*.*"
MIN_FILE_SIZE     0
RENAME_FILE       YES
RENAME_FILESPEC  "/export/home/ control/indesca/soap_listener/${origname}.${origext}"

END_EVENT
```

### Event Definition Options

The Event Definition options used in this example are:

Option	Description
EVENT_ID	Identifier that uniquely identifies an event definition record.
EVENT_TYPE	Type of system event represented by the event definition record.
COMP_NAME	Event-driven UEM Server responsible for monitoring the event.
STATE	Event definitions that should be processed or ignored by UEM.
TRACKING_INT	Event definitions that should be processed or ignored by UEM.
TRIGGERED_ID	ID of an event handler record that UEM will execute when an event occurrence is triggered.

FILESPEC	Name of a file to monitor.
MIN_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_FILESPEC	Specifies how a file should be renamed when an event occurrence is triggered.

### Loading the Event Definition

The event definition is loaded to Universal Event Monitor using the following command issued on the Linux server running Universal Command Agent for SOA.

```
/opt/universal/bin/uemload -add -deffile event_definition.txt
```

### Command Line Options

The Event Definition options used in this example are:

Option	Description
-add	Writes one or more new event definition and/or event handler records to the appropriate database.
-deffile	Name of a file that contains event definition and/or event handler parameters.

### Changing the Event Definition

Alternatively, changes to the event definition can be effected using the following command:

```
/opt/universal/bin/uemload -update -deffile event_definition.txt
```

### Command Line Options

The Event Definition options used in this example are:

Option	Description
-update	Changes one or more existing event definition and/or event handler records.
-deffile	Name of a file that contains event definition and/or event handler parameters.

### Inbound SOAP Request – Universal Event Monitor Handler Definition

The event definition 'moves' each **Process\_%Seq\$.xml** file to a staging directory and invokes a SOA HANDLER.

The following Universal Event Monitor handler definition processes each **Process\_%Seq%.xml** file.

```

BEGIN_HANDLER
HANDLER_ID      "ABC SOA HANDLER"
ACTION_TYPE     CMD
MAXRC           0
USERID          "control"
PWD             "UACL"
BEGIN_SCRIPT
  STMT "#!/usr/bin/ksh"
  STMT "exec > /export/home/control/indesca/abc.log 2>&1"
  STMT "set -xv"

  STMT "/opt/universal/bin/ucmd -script /export/home/control/indesca/abc.rexx \"
  STMT "< $UEMRENAMEDFILE \"
  STMT "-HOST mvstcp5 -USERID CTLMNT -PWD UACL "
  STMT ">> /export/home/control/indesca/abc.log \"
  STMT "2>&1"
  STMT "if [ $? -gt 0 ]"
  STMT " then"
  STMT " mv $UEMRENAMEDFILE $UEMORIGFILE"
  STMT " else"
  STMT " rm $UEMRENAMEDFILE"
  STMT " fi"
  STMT "exit $rc"
END_SCRIPT
END_HANDLER

```

The Event Handler executes under the authority of the USERID control. To allow this userid to authenticate without a password, the following UACL definitions were made to **/etc/universal/uacl.conf**:

- uem\_handler control,allow,noauth

Changes to the configuration files require the Universal Broker to be refreshed (see [Configuration Refresh](#)).

The Event Handler invokes Universal Command to:

1. Connect to the z/OS mainframe.
2. Execute a REXX script to parse the required information from the **process\_%Seq%.xml** file.
3. Execute the Universal Write-to-Operator utility to write the required console message.

The Event Handler appends logging information to the following file: **/export/home/control/indesca/abc.log**.

If the Event Handler does not complete successfully, the **process\_%Seq%.xml** file is moved back its original location so that processing can be retried. Otherwise, this file is deleted.

### Outbound SOAP Request – abc.rexx

The REXX script executed by the Event Handler is stored on the Linux server running Universal Command Agent for SOA.

```

/* REXX */
TRACE R
ABC.XML = LINEIN()

parse value ABC.XML with "<NS1:activityAction>" ABC.ACTN "</NS1:activityAction>"

parse value ABC.XML with "<NS1:identitySourceApplicationId>" ABC.APID
"</NS1:identitySourceApplicationId>"

parse value ABC.XML with "<NS1:activityRequestId>" ABC.RQID "</NS1:activityRequestId>"

ABC.UWTO = "EIEOSRAT "ABC.ACTN ABC.APID ABC.RQID

'/usr/lpp/universal/bin/uwto \-msg "'ABC.UWTO'"'
ABC.RC = RC

EXIT ABC.RC

```

The REXX script is executed under the z/OS USS environment under the authority of the USERID **CTLMNT**. To allow this userid to authenticate without a password, the following UACL definitions were made to **TEST.SYS5.UNV.UNVCONF (ACLCFG00)**:

- ucmd\_access ALL,\*,CTLMNT,allow,noauth

Changes to the configuration files require the Universal Broker to be refreshed (see [Configuration Refresh](#)).

The REXX script executes the Universal Write-to-Operator utility in order to write the required message to the z/OS console.

The **abc.log** file is appended to each time a `process_%Seq%.xml` is processed. This file is useful as an audit trail and for problem diagnosis.

### Outbound SOAP Request – Event and Handler to purge abc.log

In order to ensure that this file does not grow to an unreasonable size, additional Universal Event Monitor Event and Handler have been implemented to purge this file when it reaches 10mb in size.

```

BEGIN_EVENT
  EVENT_ID           "ABC LOG FILE CLEANUP"
  EVENT_TYPE        FILE
  COMP_NAME         UEMS
  STATE             ENABLE
  TRACKING_INT      10
  TRIGGERED_ID     "ABC LOG FILE CLEANUP"
  FILESPEC          "/export/home/control/indesca/abc.log"
  MIN_FILE_SIZE     10M
END_EVENT

BEGIN_HANDLER
  HANDLER_ID        "ABC LOG FILE CLEANUP"
  HANDLER_TYPE      CMD
  MAXRC            0
  USERID           "control"
  PWD              "UACL"
  CMD              "rm /export/home/control/indesca/abc.log"
END_HANDLER
    
```

### Event Options

The Event options used in this example are:

Option	Description
EVENT_ID	Identifier that uniquely identifies an event definition record.
EVENT_TYPE	Type of system event represented by the event definition record.
COMP_NAME	Event-driven UEM Server responsible for monitoring the event.
STATE	Event definitions that should be processed or ignored by UEM.
TRACKING_INT	Event definitions that should be processed or ignored by UEM.
TRIGGERED_ID	ID of an event handler record that UEM will execute when an event occurrence is triggered.
FILESPEC	Name of a file to monitor.
MIN_FILE_SIZE	Size a file must be in order to be considered complete by UEM.
HANDLER_ID	Identifier that uniquely identifies an event handler record.

HANDLER_TYPE	Type of process executed for the event handler.
MAXRC	Highest value with which a handler can exit to still be considered as having executed successfully.
USERID	ID of a user account in whose security context the handler process will be executed.
PWD	Password for the user account specified by userid.
CMD	Command to execute on behalf of the event handler.

## Components

Universal Event Monitor

UEMLoad

Universal Event Monitor for SOA

Universal Broker

Universal Write-to-Operator

## **Infitran - Event Monitoring and File Triggering**



## Event Monitoring and File Triggering - Overview

### Overview

The Event Monitoring and File Triggering feature of Infitran 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.

It allows one or more system events to be monitored at any given time.

The methods available for defining an event and its associated actions are described in these pages.

### Event Monitoring and File Triggering Examples

The Event Monitoring and File Triggering Examples for z/OS, Windows, and UNIX are specific to the operating systems supported by Workload Automation for the Event Monitoring and File Triggering feature of Infitran.

Links to detailed technical information on appropriate Infitran components are provided for each example.

### Universal Event Monitor Examples

The examples utilizing Universal Event Monitor 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.

## **Event Monitoring and File Triggering - Universal Event Monitor**

## Event Monitoring and File Triggering - Universal Event Monitor - Overview

### Universal Event Monitor

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 (**uemsvr**), 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.

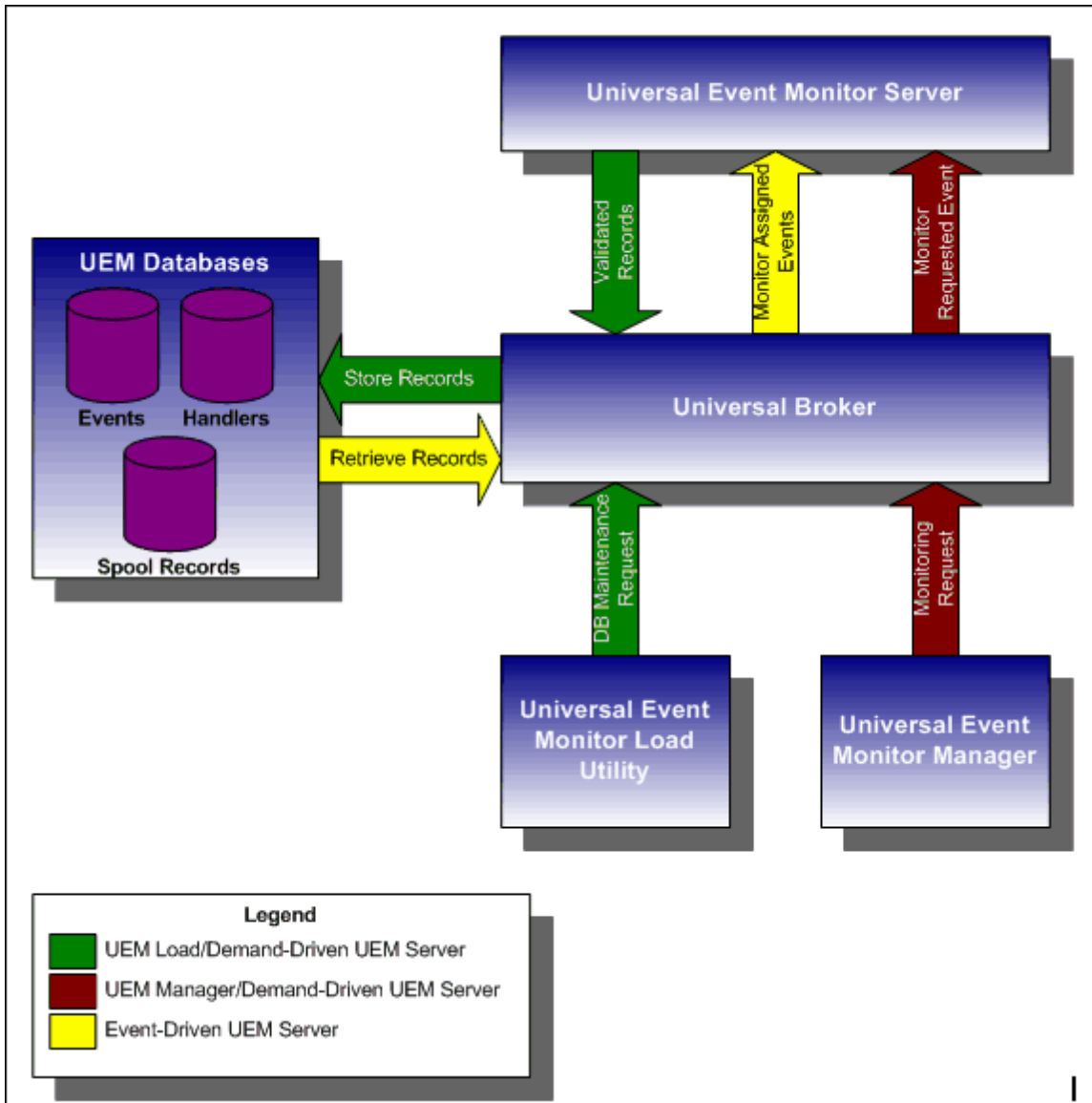
A UEM Server may monitor several local system events simultaneously using records stored in its event definition database. An event-driven UEM Server executes in this manner. An event-driven UEM Server does not require a UEM Manager to initiate a monitoring request, and you may configure it to start 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.

### High-Level Interaction of UEM Components

The following figure illustrates the interaction of the various components that make up Universal Event Monitor.



## Storing Event Definitions and Event Handlers

### 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.

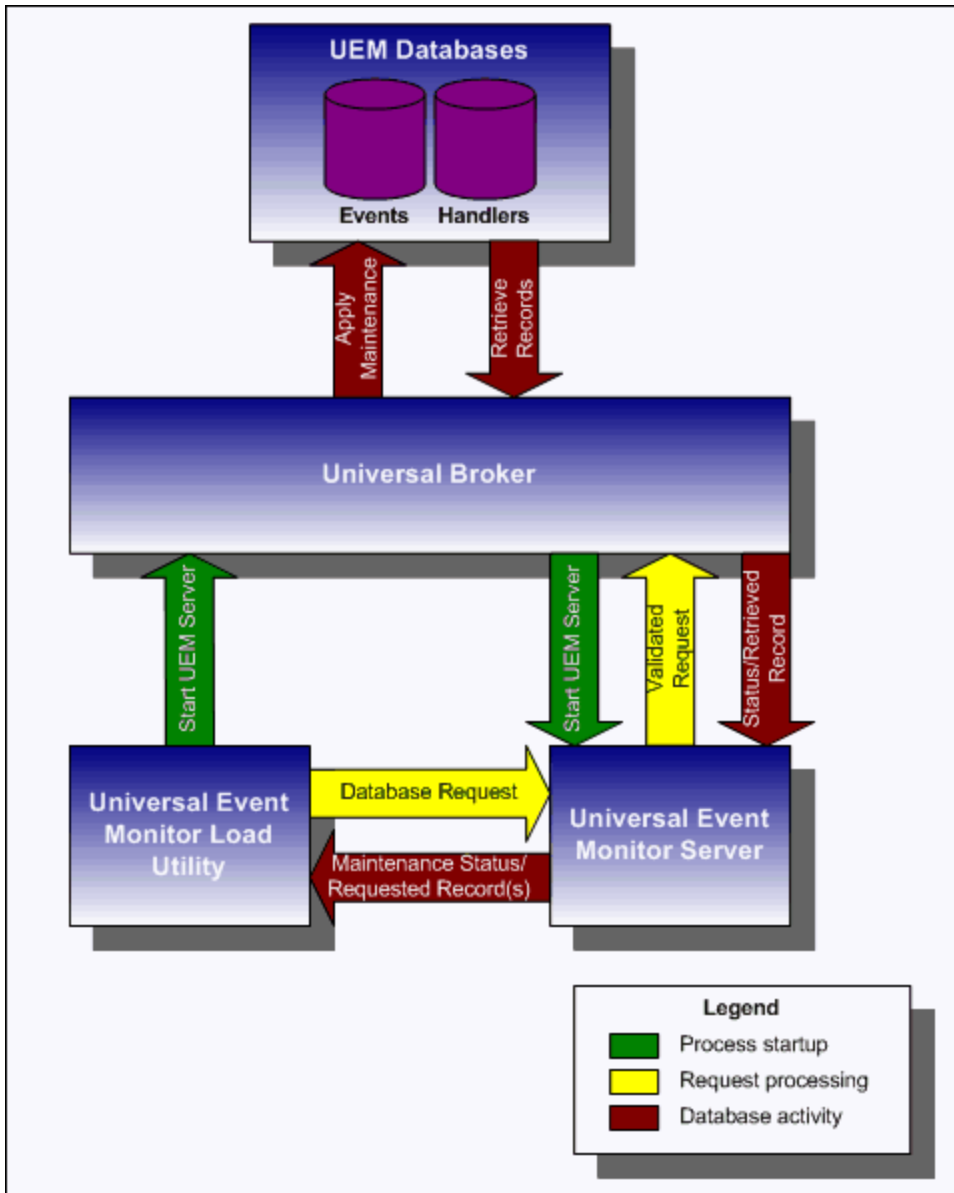
UEMLoad forwards the database request to the UEM Server, which validates it and supplies default values for any required parameters (based upon the type of request) that were not specified from the UEMLoad command line. When a set of complete, valid parameters is available, the UEM Server sends a request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

### Interaction of Universal Broker and UEM Server during UEMLoad Execution

The following figure illustrates the interaction of the Universal Broker and the Universal Event Monitor Server components involved during the execution of UEMLoad.



## Universal Event Monitor - Monitoring a Single Event

### 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* UEM 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:

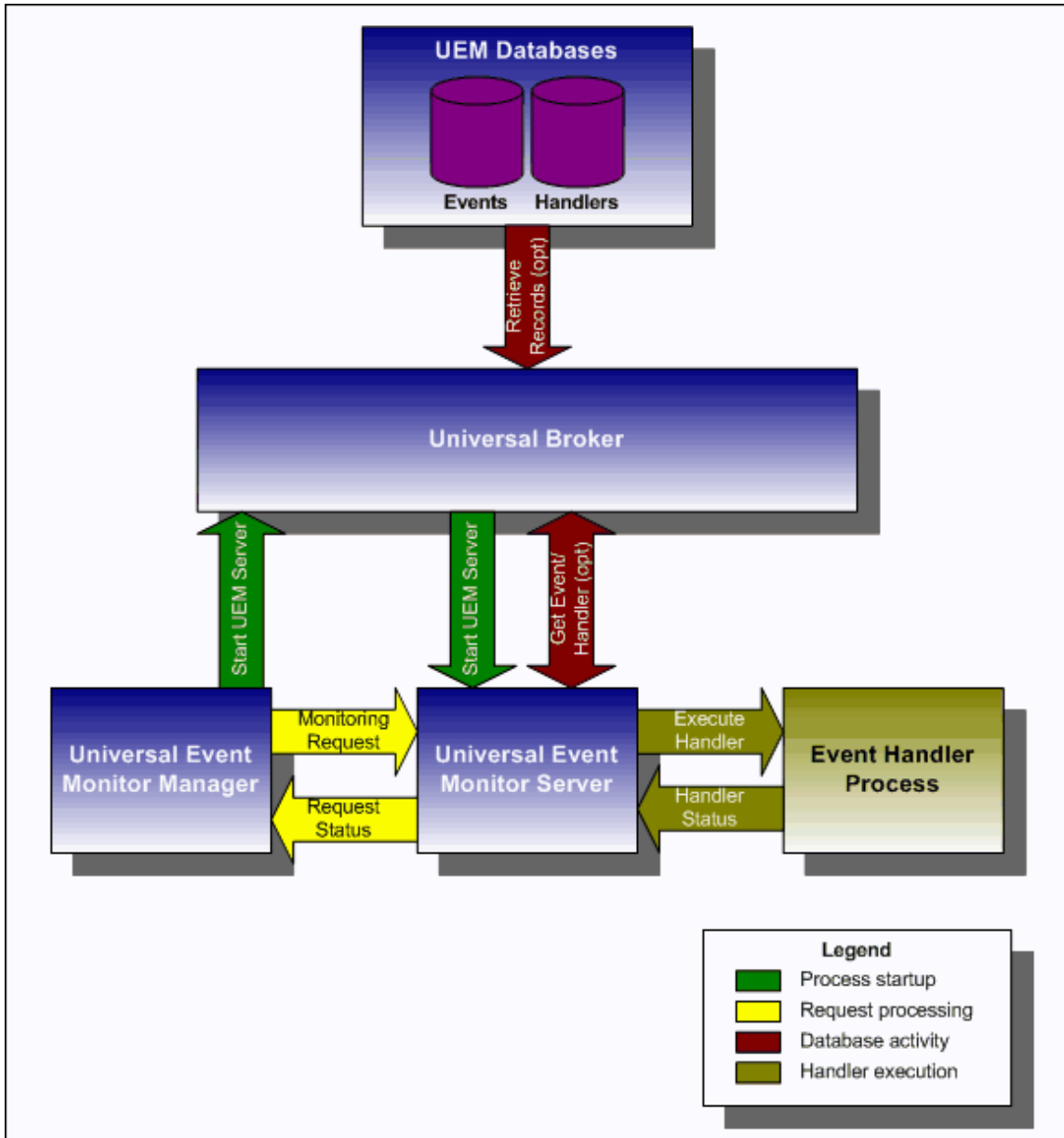
- Required number of expected event occurrences has been detected
- Inactive date and time specified for the event definition elapses.

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.

### Interaction of Universal Broker and UEM Components during UEM Manager Execution

The following figure illustrates the interaction of the Universal Broker and the Universal Event Monitor components involved when a UEM Manager is executed.





## Universal Event Monitor - Monitoring Multiple Events

### Monitoring Multiple Events

An *event-driven* UEM 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.

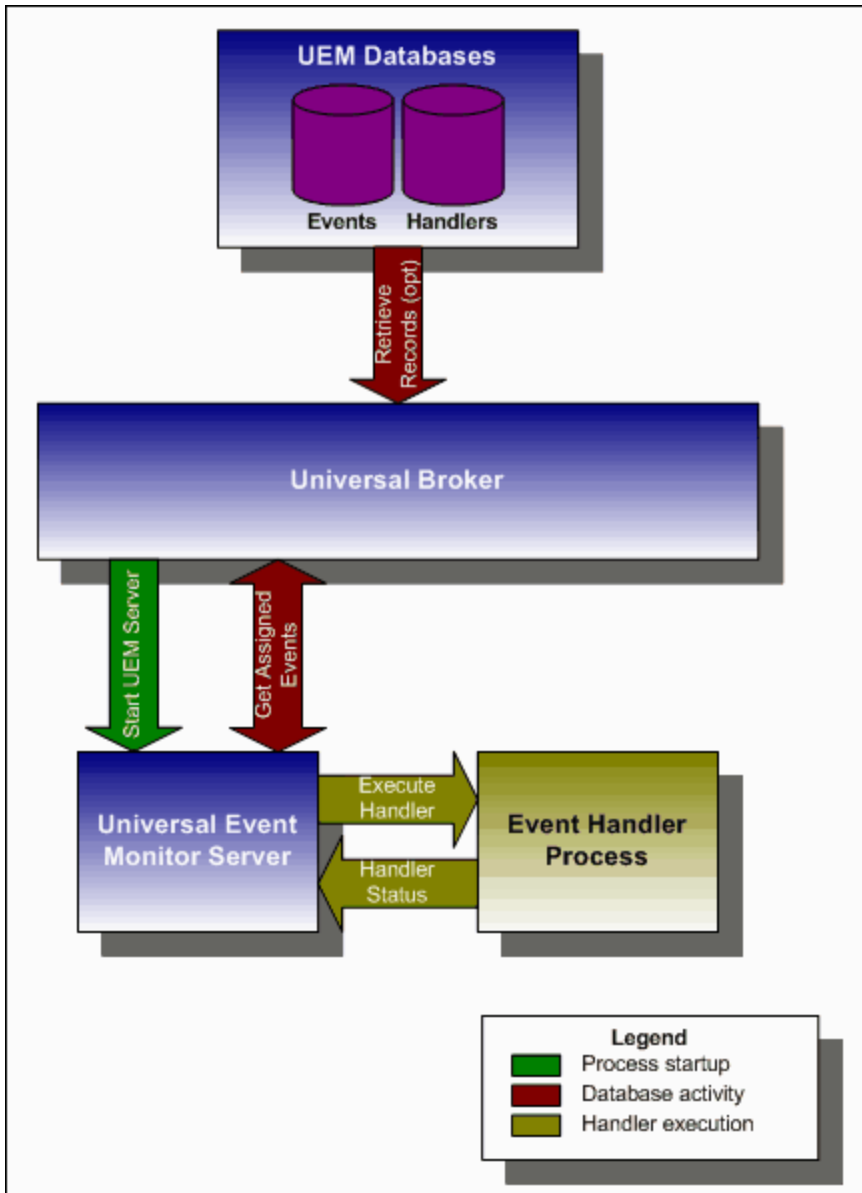
Unless it is stopped manually (using Universal Control), the event-driven UEM Server will continue to run as long as the Broker remains active. When the Broker stops, it will send a stop request to the UEM Server, instructing it to shut itself down.

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.

### Interaction of Universal Broker and an Event-Driven UEM Server

The following figure illustrates the interaction of the Universal Broker and an event-driven UEM Server.



## Event Monitoring and File Triggering - UEMLoad

- [Overview](#)
- [Controlling Database Access](#)
  - [Access via UEMLoad Utility](#)
  - [Universal Access Control List](#)

### 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 [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.

For detailed information on the event definition and event handler database files, see [UEMLoad Utility](#).

### Controlling Database Access

Universal Broker is primarily responsible for providing access to the Workload Automation 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 [Workload Automation Utilities 5.1.0 Reference 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 [UEM Server Database Files](#).

### 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.

## **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.

## Event Monitoring and File Triggering - Examples

- [Event Monitoring and File Triggering Examples - z/OS](#)
- [Event Monitoring and File Triggering Examples - Windows](#)
- [Event Monitoring and File Triggering Examples - UNIX](#)

### Event Monitoring and File Triggering Examples - z/OS

- [Starting an Event-Driven UEM Server - z/OS](#)
- [Refreshing an Event-Driven UEM Server - z/OS](#)
- [Using a Stored Event Handler Record - z/OS](#)
- [Handling an Event with a Script - z/OS](#)
- [Handling an Expired Event - z/OS](#)
- [Continuation Character \( - \) in z/OS Handler Script](#)
- [Continuation Character \( + \) in z/OS Handler Script](#)
- [Continuation Characters \( - and + \) in z/OS Handler Script](#)

### Event Monitoring and File Triggering Examples - Windows

- [Using a Stored Event Handler Record - Windows](#)
- [Execute Script for Triggered Event Occurrence - Windows](#)
- [Handling an Expired Event - Windows](#)
- [Add a Single Event Record - Windows](#)
- [Add a Single Event Handler Record - Windows](#)
- [List All Event Definitions - Windows](#)
- [Export Event Definition and Handler Databases - Windows](#)
- [List a Single Event Handler Record - Windows](#)
- [List Event Definitions and Handlers Using Wildcards - Windows](#)
- [Add Record\(s\) Using Definition File - Windows](#)
- [Add Records Remotely Redirected from STDIN - Windows](#)
- [Add Records Redirected from STDIN \(for zOS\) - Windows](#)
- [Definition File Format - Windows](#)

### Event Monitoring and File Triggering Examples - UNIX

- [Using a Stored Event Handler Record - UNIX](#)
- [Execute Script for Triggered Event Occurrence - UNIX](#)
- [Handling an Expired Event - UNIX](#)
- [Add a Single Event Record - UNIX](#)
- [Add a Single Event Handler Record - UNIX](#)
- [List All Event Definitions for UNIX](#)
- [Export Event Definition and Handler Databases - UNIX](#)
- [List a Single Event Handler Record for UNIX](#)
- [List Event Definitions and Handlers Using Wildcards for UNIX](#)
- [Add Record\(s\) Using Definition File for UNIX](#)
- [Add Record\(s\) Remotely Redirected from STDIN - UNIX](#)
- [Add Record\(s\) Remotely Redirected from STDIN \(for zOS\) - UNIX](#)
- [Definition File Format - UNIX](#)

## Starting an Event-Driven UEM Server - zOS

### Starting an Event-Driven UEM Server

There are two ways start an event-driven UEM Server (**uems**) component:

1. Recycle the **ubroker** daemon (Universal 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 -encryptedfile LOGONDD -port 7887 -start uems
/*
```



#### Note

There is only one different command (**-start**) between this example and [Refreshing an Event-Driven UEM Server](#).

### SYSIN Options

The SYSIN options used in this example are:

Option	Description
-host	TCP/IP host name of the remote computer on which Universal Broker is running and accepting connections.
-encrypted	Encrypted command file.
-port	TCP/IP port number of the remote computer on which Universal Broker is running and accepting connections.
-start	Instruction to a Universal Broker to start the UEM Server.

### Components

[Universal Control](#)

[Universal Event Monitor Server for Windows](#)

[Universal Event Monitor Server for UNIX](#)

## Refreshing an Event-Driven UEM Server - zOS

### Refreshing an Event-Driven UEM Server

In this example, RESUEMS will refresh the event-driven UEM 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 -encryptedfile LOGONDD -port 7887 -refresh uems
/*
```



#### Note

There is only one different command (**-refresh**) between this example and [Starting an Event-Driven UEM Server](#).

### SYSIN Options

The SYSIN options used in this example are:

Option	Description
-host	TCP/IP host name of the remote computer on which Universal Broker is running and accepting connections.
-encrypted	Encrypted command file.
-port	TCP/IP port number of the remote computer on which Universal Broker is running and accepting connections.
-refresh	Instruction to the Universal Broker to refresh the UEM Server configuration.

### Components

Universal Control

Universal Event Monitor Server for Windows

Universal Event Monitor Server for UNIX

## Using a Stored Event Handler Record - zOS

### Using a Stored Event Handler Record in z/OS

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
-handler_id h001
-host uemhost
-userid uemuser
-pwd uemusers_password
-max_count 1
/*
```

### SYSIN Options

The SYSIN options used in this example are:

Option	Description
<code>-event_type</code>	Type of event to monitor.
<code>-filespec</code>	Name or pattern of the file whose creation should be detected and tracked for completion.
<code>-wait</code>	Forces the UEM Manager to wait for the completion of the UEM Server.
<code>-inact_date_time</code>	Date and time at which the state of the monitored event should be made inactive.
<code>-triggered</code>	Event state that, when encountered, will result in the execution of the associated event handler.
<code>-handler_id</code>	ID of a stored event handler record.



<code>-host</code>	List of one or more hosts upon which a command may run.
<code>-userid</code>	ID of a remote user account that the UEM Server uses to establish the security context in which event monitoring is performed.
<code>-pwd</code>	Password associated with <code>-userid</code> .
<code>-max_count</code>	Maximum number of event occurrences to monitor.

## Components

Universal Event Monitor Manager for z/OS

Universal Event Monitor Server

## Handling an Event with a Script - zOS

### Handling an Event With a Script in z/OS

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 myscrip
/*
```

### SYSIN Options

The SYSIN options used in this example are:

Option	Description
<code>-event_type</code>	Type of event to monitor.

<code>-filespec</code>	Name or pattern of the file whose creation should be detected and tracked for completion.
<code>-inact_date_time</code>	Date and time at which the state of the monitored event should be made inactive.
<code>-handler_opts</code>	Forces the UEM Manager to wait for the completion of the UEM Server.
<code>-host</code>	List of one or more hosts upon which a command may run.
<code>-userid</code>	ID of a remote user account that the UEM Server uses to establish the security context in which event monitoring is performed.
<code>-pwd</code>	Password associated with <code>-userid</code> .
<code>-triggered</code>	Event state that, when encountered, will result in the execution of the associated event handler.
<code>-script</code>	Complete path to a local script file or DD statement that contains one or more system commands that should be executed on behalf of the event handler.

## Components

[Universal Event Monitor Manager for z/OS](#)

[Universal Event Monitor Server for Windows](#)

## Handling an Expired Event - zOS

### Handling an Expired Event in z/OS

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

In this example, the **-expired** option is followed by the **-options** option, which redirects the output of the "ls -alR /uem files" command to a file in uemuser's home directory called **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
/*
```

### SYSIN Options

The SYSIN options used in this example are:

Option	Description
<code>-event_type</code>	Type of event to monitor.
<code>-filespec</code>	Name or pattern of the file whose creation should be detected and tracked for completion.
<code>-inact_date_time</code>	Date and time at which the state of the monitored event should be made inactive.
<code>-expired</code>	Event state that, when encountered, will result in the execution of the associated event handler.
<code>-cmd</code>	Complete path to an application file or remote script that should be executed on behalf of the event handler.

<code>-host</code>	List of one or more hosts upon which a command may run.
<code>-userid</code>	ID of a remote user account that the UEM Server uses to establish the security context in which event monitoring is performed.
<code>-pwd</code>	Password associated with <code>-userid</code> .

## Components

Universal Event Monitor Manager for z/OS

Universal Event Monitor Server for UNIX

## Continuation Character ( - ) in zOS Handler Script

### 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.....: 06/11/11 10:32:31 AM
Last Modified By.....: mfc1a
```

## Components

[Universal Event Monitor Manager for z/OS](#)

## Continuation Character ( + ) in zOS Handler Script

### 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.....: 06/11/11 11:46:32 AM
Last Modified By.....: mfc1a
```

## Components

Universal Event Monitor Manager for z/OS

## Continuation Characters ( - and + ) in zOS Handler Script

### 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 - character will preserve trailing spaces in your line. The + 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.....: 06/11/1 01:25:20 PM
Last Modified By.....: mfc1a
```

## Components

Universal Event Monitor Manager for z/OS



## Using a Stored Event Handler Record - Windows

### Using a Stored Event Handler Record in Windows

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
```

### Command Line Options

The command line options used in this example are:

Option	Description
-host	List of one or more hosts upon which a command may run.
-event_type	Type of event to monitor.
-filespec	Name or pattern of the file whose creation should be detected and tracked for completion.
-inact_date_time	Date and time at which the state of the monitored event should be made inactive.
-userid	ID of a remote user account that the UEM Server uses to establish the security context in which event monitoring is performed.
-pwd	Password associated with -userid.
-triggered	Event state that, when encountered, will result in the execution of the associated event handler.
-handler_id	ID of a stored event handler record.

## **Components**

Universal Event Monitor Manager for Windows

## Execute Script for Triggered Event Occurrence - Windows

- Executing a Script for a Triggered Event Occurrence in Windows
  - Command Line Options
  - Contents of Sample Script File
  - Components

### Executing a Script for a Triggered Event Occurrence in Windows

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"
```

### Command Line Options

The command line options used in this example are:

Option	Description
<b>-host</b>	List of one or more hosts upon which a command may run.
<b>-event_type</b>	Type of event to monitor.
<b>-filespec</b>	Name or pattern of the file whose creation should be detected and tracked for completion.
<b>-inact_date_time</b>	Date and time at which the state of the monitored event should be made inactive.
<b>-userid</b>	ID of a remote user account that the UEM Server uses to establish the security context in which event monitoring is performed.
<b>-pwd</b>	Password associated with <b>-userid</b> .
<b>-triggered</b>	Event state that, when encountered, will result in the execution of the associated event handler.

<code>-script</code>	Complete path to a local script file that contains one or more system commands that should be executed on behalf of the event handler.
<code>-handler_opts</code>	Options that are passed as command line arguments to any process executed on behalf of an event handler.

### Contents of Sample Script File

The following figure 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
```

### Components

Universal Event Monitor Manager for Windows

Universal Event Monitor Server for UNIX

## Handling an Expired Event - Windows

### Handling an Expired Event in Windows

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** option is necessary to accommodate parsing requirements for command options in Windows (see the UEM Manager [FILE\\_SPECIFICATION](#) option).

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

In this example, the **-expired** option is followed by the **-options** option, which 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"
```

### Command Line Options

The command line options used in this example are:

Option	Description
<b>-host</b>	List of one or more hosts upon which a command may run.
<b>-event_type</b>	Type of event to monitor.
<b>-userid</b>	ID of a remote user account that the UEM Server uses to establish the security context in which event monitoring is performed.
<b>-pwd</b>	Password associated with <b>-userid</b> .
<b>-filespec</b>	Name or pattern of the file whose creation should be detected and tracked for completion.

<code>-inact_date_time</code>	Date and time at which the state of the monitored event should be made inactive.
<code>-expired</code>	Event state that, when encountered, will result in the execution of the associated event handler.
<code>-cmd</code>	Complete path to an application file or remote script that should be executed on behalf of the event handler.
<code>-options</code>	Values that are passed as command line arguments to a particular handler specified for a given <code>EVENT_STATE</code> .

## Components

Universal Event Monitor Manager for Windows

Universal Event Monitor Server for UNIX

## Add a Single Event Record - Windows

### Adding a Single Event Record for Windows

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 UEMLoad `EVENT_STATE`, `ACTIVE_DATE_TIME`, and `INACTIVE_DATE_TIME` options were specified, the default value of **enable**, 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
```

### Command Line Options

The command line options used in this example are:

Option	Description
<code>-add</code>	Writes one or more new event definition and/or event handler records to the appropriate database.
<code>-event_id</code>	Identifier that uniquely identifies an event definition record.
<code>-event_type</code>	Type of system event represented by the event definition record.
<code>-filespec</code>	Name of a file to monitor.
<code>-triggered_id</code>	ID of an event handler record that UEM will execute when an event occurrence is triggered.

### Components

UEMLoad Utility for Windows

Universal Event Monitor Server for UNIX

## Add a Single Event Handler Record - Windows

### Adding a Single Event Handler Record for Windows

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** option, 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"
```

### Command Line Options

The command line options used in this example are:

Option	Description
<code>-add</code>	Writes one or more new event definition and/or event handler records to the appropriate database.
<code>-handler_id</code>	Identifier that uniquely identifies an event handler record.
<code>-encryptedfile</code>	Complete path to a file encrypted with Universal Encrypt.
<code>-cmd</code>	Command to execute on behalf of the event handler.

### Components

[UEMLoad Utility for Windows](#)

[Universal Event Monitor Server for UNIX](#)

[Universal Encrypt](#)



## List All Event Definitions - Windows

### Listing All Event Definitions for Windows

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 *
```



#### Note

The default behavior when listing or exporting records, when neither an event ID nor a handler ID is specified, is to return all records. However, in this example, 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.

### Command Line Options

The command line options used in this example are:

Option	Description
-list	Displays the complete contents of the specified event definition and/or event handler records.
-event_id	Identifier that uniquely identifies an event definition record.

### Components

UEMLoad Utility for Windows

## Export Event Definition and Handler Databases - Windows

### Exporting the Event Definition and Event Handler Databases for Windows

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 example shown in [Definition File Format - Windows](#).

```
-export -deffile uemout.txt
```



#### Note

No event ID or handler ID is specified from the command line. If neither parameter is specified when listing or exporting records, the default behavior is to retrieve all database records.

### Command Line Options

The command line options used in this example are:

Option	Description
<code>-export</code>	Dumps the contents of the specified event definition and/or event handler records to a text file that can be used as input to a subsequent run of the UEMLoad utility.
<code>-deffile</code>	Name of a file that contains event definition and/or event handler parameters.

### Components

[UEMLoad Utility for Windows](#)

## List a Single Event Handler Record - Windows

### List a Single Event Handler Record for Windows

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
```

The following figure 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 [Add a Single Event Handler Record - Windows](#).)

In this specific instance, the user ID contained in **encrypted.file** (from [Add a Single Event Handler Record - Windows](#)) is **sparkie**, and the record was added by the user account with an ID of **sbuser**.

### Command Line Options

The command line options used in this example are:

Option	Description
-list	Displays the complete contents of the specified event definition and/or event handler records.
-handler_id	Identifier that uniquely identifies an event handler record.

### Sample List Output

```
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 05/25/2011.

Event Handler(s):
=====
Handler ID.....: dirlist
Max Acceptable Return Code.: 0
User ID.....: sparkie
Command.....: ls -al
Last Modified On.....: 05/25/2011 11:32:06 AM
Last Modified By.....: sbuser

UNV3667I Universal Event Monitor Load is ending successfully with exit code 0.
```

### Components

[UEMLoad Utility for Windows](#)

## List Event Definitions and Handlers Using Wildcards - Windows

### Listing Multiple Event Definitions and Event Handlers Using Wildcards for Windows

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??
```

### Command Line Options

The command line options used in this example are:

Option	Description
-list	Displays the complete contents of the specified event definition and/or event handler records.
-event_id	Identifier that uniquely identifies an event definition record.
-handler_id	Identifier that uniquely identifies an event handler record.

### Components

UEMLoad Utility for Windows

## Add Record(s) Using Definition File - Windows

### Add Record(s) Using a Definition File for Windows

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 [Definition File Format - Windows](#).

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
```

### Command Line Options

The command line options used in this example are:

Option	Description
<a href="#">-add</a>	Writes one or more new event definition and/or event handler records to the appropriate database.
<a href="#">-deffile</a>	Name of a file that contains event definition and/or event handler parameters.

### Components

[UEMLoad Utility for Windows](#)

## Add Records Remotely Redirected from STDIN - Windows

### Add Record(s) Remotely, Using a Definition File Redirected from STDIN for Windows

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 [Definition File Format - Windows](#).

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 rmt host -encryptedfile rmtacctinfo.enc <uemadd.txt
```

### Command Line Options

The command line options used in this example are:

Option	Description
-cmd	Remote command to execute.
-add	Writes one or more new event definition and/or event handler records to the appropriate database.
-host	List of one or more hosts upon which a command may run.
-encryptedfile	Encrypted command file.

### Components

[UEMLoad Utility for Windows](#)

[Universal Command Manager for Windows](#)

[Universal Event Monitor Server](#)

## Add Records Redirected from STDIN (for zOS) - Windows

### Add Record(s) Remotely, Using a Definition File Redirected from STDIN (for z/OS) for Windows

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 [Definition File Format - Windows](#).

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"
```

### SYSIN Options

The SYSIN options used in this example are:

Option	Description
-host	List of one or more hosts upon which a command may run.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-cmd	Remote command to execute.
-add	Writes one or more new event definition and/or event handler records to the appropriate database.

### Components

[UEMLoad Utility for Windows](#)

[Universal Command Manager for zOS](#)

[Universal Event Monitor Server](#)

## Definition File Format - Windows

### Definition File Format for Windows

The format of the definition load file for events and event handlers follows the keyword / value-pair convention used for Workload Automation 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 [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 the following figure.



```

# 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 2011.12.31,23:59
  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".

```

### Definition File Options

The Definition File options used in this example are:

Option	Description
<a href="#">event_id</a>	Identifier that uniquely identifies an event definition record.
<a href="#">event_type</a>	Type of system event represented by the event definition record.
<a href="#">comp_name</a>	Event-driven UEM Server responsible for monitoring the event.
<a href="#">state</a>	Event definitions that should be processed or ignored by UEM.

<code>inact_date_time</code>	Date and time at which UEM will stop monitoring an event definition.
<code>triggered_id</code>	ID of an event handler record that UEM will execute when an event occurrence is triggered.
<code>filespec</code>	Name of a file to monitor.
<code>rename_file</code>	Specification for whether or not UEM should rename a monitored file when an event occurrence is triggered.
<code>rename_filespec</code>	Specification for how a file should be renamed when an event occurrence is triggered.
<code>handler_id</code>	Identifier that uniquely identifies an event handler record.
<code>handler_type</code>	Type of process executed for the event handler, based on the contents of the <code>USER_COMMAND</code> and <code>USER_SCRIPT</code> parameters.
<code>maxrc</code>	Highest value with which a handler can exit to still be considered as having executed successfully.
<code>userid</code>	ID of a user account in whose security context the handler process will be executed.
<code>script_type</code>	Type of script statements contained in the action field of the event handler record.
<code>cmd</code>	Command to execute on behalf of the event handler.

## Components

UEMLoad Utility for Windows

## Using a Stored Event Handler Record - UNIX

### Using a Stored Event Handler Record in UNIX

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
```

### Command Line Options

The command line options used in this example are:

Option	Description
-host	List of one or more hosts upon which a command may run.
-event_type	Type of event to monitor.
-filespec	Name or pattern of the file whose creation should be detected and tracked for completion.
-inact_date_time	Date and time at which the state of the monitored event should be made inactive.
-userid	ID of a remote user account that the UEM Server uses to establish the security context in which event monitoring is performed.
-pwd	Password associated with -userid.
-triggered	Event state that, when encountered, will result in the execution of the associated event handler.
-handler_id	ID of a stored event handler record.

## **Components**

Universal Event Monitor Manager for UNIX

Universal Event Monitor Server for Windows

## Execute Script for Triggered Event Occurrence - UNIX

- Executing a Script for a Triggered Event Occurrence in UNIX
  - Command Line Options
  - Contents of Sample Script File
  - Components

### Executing a Script for a Triggered Event Occurrence in UNIX

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"
```

### Command Line Options

The command line options used in this example are:

Option	Description
<code>-host</code>	List of one or more hosts upon which a command may run.
<code>-event_type</code>	Type of event to monitor.
<code>-filespec</code>	Name or pattern of the file whose creation should be detected and tracked for completion.
<code>-inact_date_time</code>	Date and time at which the state of the monitored event should be made inactive.
<code>-userid</code>	ID of a remote user account that the UEM Server uses to establish the security context in which event monitoring is performed.
<code>-pwd</code>	Password associated with <code>-userid</code> .
<code>-triggered</code>	Event state that, when encountered, will result in the execution of the associated event handler.

<code>-script</code>	Complete path to a local script file that contains one or more system commands that should be executed on behalf of the event handler.
<code>-handler_opts</code>	Options that are passed as command line arguments to any process executed on behalf of an event handler.

### Contents of Sample Script File

The following figure 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

```

### Components

Universal Event Monitor Manager for UNIX

Universal Event Monitor Server for Windows

## Handling an Expired Event - UNIX

### Handling an Expired Event in UNIX

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 the home directory of **uemuser**.

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** option corresponding to the **-expired** option. In this example, UEM executes the **'ls -alR /uem files'** command.



#### Note

In this example, the **-expired** option is followed by the **-options** option, which redirects the output of the **'ls -alR "/uем 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 "/uем files"' -options '>uemtest.log 2>&1'
```

### Command Line Options

The command line options used in this example are:

Option	Description
<b>-host</b>	List of one or more hosts upon which a command may run.
<b>-event_type</b>	Type of event to monitor.
<b>-filespec</b>	Name or pattern of the file whose creation should be detected and tracked for completion.
<b>-userid</b>	ID of a remote user account that the UEM Server uses to establish the security context in which event monitoring is performed.
<b>-pwd</b>	Password associated with <b>-userid</b> .
<b>-inact_date_time</b>	Date and time at which the state of the monitored event should be made inactive.
<b>-expired</b>	Event state that, when encountered, will result in the execution of the associated event handler.

<code>-cmd</code>	Complete path to an application file or remote script that should be executed on behalf of the event handler.
<code>-options</code>	Values that are passed as command line arguments to a particular handler specified for a given <code>EVENT_STATE</code> .

## Components

Universal Event Monitor Manager for UNIX

Universal Event Monitor Server for UNIX



## Add a Single Event Record - UNIX

### Adding a Single Event Record for UNIX

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 UEMLoad `EVENT_STATE`, `ACTIVE_DATE_TIME`, and `INACTIVE_DATE_TIME` options were specified, the default values of **enable**, 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
```

### Command Line Options

The command line options used in this example are:

Option	Description
<code>-add</code>	Writes one or more new event definition and/or event handler records to the appropriate database.
<code>-event_id</code>	Identifier that uniquely identifies an event definition record.
<code>-event_type</code>	Type of system event represented by the event definition record.
<code>-filespec</code>	Name of a file to monitor.
<code>-triggered_id</code>	ID of an event handler record that UEM will execute when an event occurrence is triggered.

### Components

UEMLoad Utility for UNIX

Universal Event Monitor Server for UNIX

## Add a Single Event Handler Record - UNIX

### Adding a Single Event Handler Record for UNIX

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** option, 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 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"
```

### Command Line Options

The command line options used in this example are:

Option	Description
<code>-add</code>	Writes one or more new event definition and/or event handler records to the appropriate database.
<code>-handler_id</code>	Identifier that uniquely identifies an event handler record.
<code>-encryptedfile</code>	Complete path to a file encrypted with Universal Encrypt.
<code>-cmd</code>	Command to execute on behalf of the event handler.

### Components

UEMLoad Utility for UNIX

Universal Event Monitor Server for UNIX

Universal Encrypt

## List All Event Definitions for UNIX

### Listing All Event Definitions for UNIX

In this example, the **-list** option is used to dump all records in the event definition database and display them to **stdout**.

The asterisk ( **\*** ) must be escaped or enclosed in double quotation marks (that is: **\\*** or **""**), respectively).

```
uemload -list -event_id \*
```



#### Note

The default behavior when listing or exporting 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.

### Command Line Options

The command line options used in this example are:

Option	Description
<code>-list</code>	Displays the complete contents of the specified event definition and/or event handler records.
<code>-event_id</code>	Identifier that uniquely identifies an event definition record.

### Components

[UEMLoad Utility for UNIX](#)

## List a Single Event Handler Record for UNIX

### List a Single Event Handler Record for UNIX

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
```

#### Command Line Options

The command line options used in this example are:

Option	Description
<code>-list</code>	Displays the complete contents of the specified event definition and/or event handler records.
<code>-handler_id</code>	Identifier that uniquely identifies an event handler record.

#### Sample List Output

The following figure 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 [Add a Single Event Handler Record - UNIX](#).)

In this specific instance, the user ID contained in **encrypted.file** (from [Add a Single Event Handler Record - UNIX](#)) 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 11:32:04 AM 05/25/2011.

Event Handler(s):
=====
Handler ID.....: dirlist
Max Acceptable Return Code.: 0
User ID.....: sparkie
Command.....: ls -al
Last Modified On.....: 05/25/2011 11:32:06 AM
Last Modified By.....: sbuser

UNV3667I Universal Event Monitor Load is ending successfully with exit code 0.
```

## Components

UEMLoad Utility for UNIX

## Export Event Definition and Handler Databases - UNIX

### Exporting the Event Definition and Event Handler Databases for UNIX

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 [Definition File Format - UNIX](#).

```
uemload -export -deffile uemout.txt
```



#### Note

No event ID or handler ID is specified from the command line. If neither parameter is specified when listing or exporting records, the default behavior is to retrieve all database records.

### Command Line Options

The command line options used in this example are:

Option	Description
<code>-export</code>	Dumps the contents of the specified event definition and/or event handler records to a text file that can be used as input to a subsequent run of the UEMLoad utility.
<code>-deffile</code>	Name of a file that contains event definition and/or event handler parameters.

### Components

[UEMLoad Utility for UNIX](#)

## List Event Definitions and Handlers Using Wildcards for UNIX

### Listing Multiple Event Definitions and Event Handlers Using Wildcards for UNIX

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??
```

### Command Line Options

The command line options used in this example are:

Option	Description
-list	Displays the complete contents of the specified event definition and/or event handler records.
-event_id	Identifier that uniquely identifies an event definition record.
-handler_id	Identifier that uniquely identifies an event handler record.

### Components

[UEMLoad Utility for UNIX](#)

## Add Record(s) Using Definition File for UNIX

### Add Record(s) Using a Definition File for UNIX

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 [Definition File Format - UNIX](#).

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
```

### Command Line Options

The command line options used in this example are:

Option	Description
<code>-add</code>	Writes one or more new event definition and/or event handler records to the appropriate database.
<code>-deffile</code>	Name of a file that contains event definition and/or event handler parameters.

### Components

[UEMLoad Utility for UNIX](#)

## Add Record(s) Remotely Redirected from STDIN - UNIX

### Add Record(s) Remotely, Using a Definition File Redirected from STDIN for UNIX

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 [Definition File Format - UNIX](#).

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 rmt host -encryptedfile rmtacctinfo.enc <uemadd.txt
```

### Command Line Options

The command line options used in this example are:

Option	Description
-cmd	Remote command to execute.
-add	Writes one or more new event definition and/or event handler records to the appropriate database.
-host	List of one or more hosts upon which a command may run.
-encryptedfile	Encrypted command file.

### Components

[Universal Command Manager for UNIX](#)

[UEMLoad Utility for UNIX](#)

[Universal Event Monitor Server for UNIX](#)



## Add Record(s) Remotely Redirected from STDIN (for zOS) - UNIX

### Add Record(s) Remotely, Using a Definition File Redirected from STDIN (for z/OS) for UNIX

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 [Definition File Format - UNIX](#).

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"
/*
```

### SYSIN Options

The SYSIN options used in this example are:

Option	Description
-host	List of one or more hosts upon which a command may run.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-cmd	Remote command to execute.
-add	Writes one or more new event definition and/or event handler records to the appropriate database.

### Components

Universal Command Manager for zOS

UEMLoad Utility for UNIX

Universal Event Monitor Server for UNIX

## Definition File Format - UNIX

### Definition File Format for UNIX

The format of the definition load file for events and event handlers follows the keyword / value-pair convention used for Workload Automation 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 [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 UNIX is shown in the following figure.

```

# 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 2011.12.31,23:59
  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 'homeuem usersomeapp'
end_handler

# End of parameters for event definition "unix_cmd_sample".

```

**Definition File Options**

The Definition File options used in this example are:

Option	Description
event_id	Identifier that uniquely identifies an event definition record.
event_type	Type of system event represented by the event definition record.
comp_name	Event-driven UEM Server responsible for monitoring the event.
state	Event definitions that should be processed or ignored by UEM.

<code>inact_date_time</code>	Date and time at which UEM will stop monitoring an event definition.
<code>triggered_id</code>	ID of an event handler record that UEM will execute when an event occurrence is triggered.
<code>filespec</code>	Name of a file to monitor.
<code>rename_file</code>	Specification for whether or not UEM should rename a monitored file when an event occurrence is triggered.
<code>rename_filespec</code>	Specification for how a file should be renamed when an event occurrence is triggered.
<code>handler_id</code>	Identifier that uniquely identifies an event handler record.
<code>handler_type</code>	Type of process executed for the event handler, based on the contents of the <code>USER_COMMAND</code> and <code>USER_SCRIPT</code> parameters.
<code>maxrc</code>	Highest value with which a handler can exit to still be considered as having executed successfully.
<code>userid</code>	ID of a user account in whose security context the handler process will be executed.
<code>script_type</code>	Type of script statements contained in the action field of the event handler record.
<code>cmd</code>	Command to execute on behalf of the event handler.

## Components

UEMLoad Utility for UNIX

## Infitran - Encryption

## Encryption - Overview

- Encryption
- Encrypting Files
- Transferring Encrypted Files between Servers
  - Security Considerations

### Encryption

Workload Automation programs have the ability to read command line options contained in command files. Command files that contain private information must be protected by using local file system security. This ensures that only authorized accounts have read access.

The [Universal Encrypt \(UENCRYPT\)](#) utility adds an additional layer of security by encrypting the contents of command files into an unintelligible format.

Although all command line options can be encrypted with the Universal Encrypt utility, most organizations use it to encrypt and store authentication credentials such as user ID and/or password.

An encrypted command file can be decrypted only by Workload Automation product programs. No decrypt command is provided to decrypt the command file.



#### Note

Universal Encrypt should not be used as a replacement for file system security.

### Encrypting Files

Files do not have to be encrypted on the same platform or server on which they will be used. They can be encrypted on any platform or server and then transferred. This means that applications development, platform administrators, and security administrators can encrypt passwords in their own environments.

Universal Encrypt encrypts files with either:

- 56-bit DES
- 256-bit AES

Universal Encrypt reads an unencrypted file from its standard input and writes the encrypted version to its standard output.

Encrypted files are text files and contain comments that can be edited if required. Lines within the encrypted file that start with the # character are comments. Default comments are created with the following information:

- Date of encryption.
- Userid that encrypted the file.
- System on which the file was encrypted.
- Version of Universal Encrypt used.
- Level of encryption used.

### Transferring Encrypted Files between Servers

Files encrypted via Universal Encrypt are text files.

You can transfer them between servers, using FTP or similar tools, in text mode. You also can email them between like systems (for example, Windows to Windows).

### Security Considerations

For production implementations, thought should be given to the location and security of encrypted files containing passwords. Consider who needs access to create, update, and use these files.

Many implementations are centralized around an enterprise scheduling solution. In this case, the encrypted files are often secured in such a way that only the enterprise scheduler is able to access them.

There are additional layers of security available to Indesca, such as [Universal Access Control List](#) and [X.509 Certificates](#). These can be further used to ensure that access to servers is properly controlled.



## Encryption - Examples

### Encryption Examples

This page provide examples of how to use Universal Encrypt to encrypt a command file (and how to use the encrypted file). Each example will encrypt a case sensitive password using AES 256 encryption.

Links to detailed technical information on appropriate Infitran components are provided for each example.

### Encryption Examples - z/OS

- [Creating Encrypted Command File for z/OS](#)
- [Using Encrypted Command File on z/OS](#)

### Encryption Examples - Windows

- [Creating Encrypted Command File for Windows](#)
- [Using Encrypted Command File on Windows](#)

### Encryption Examples - UNIX

- [Creating Encrypted Command File for UNIX](#)
- [Using Encrypted Command File on UNIX](#)

### Encryption Examples - IBM i

- [Creating Encrypted Command File for IBM i](#)
- [Using Encrypted Command File on IBM i](#)



## Creating Encrypted Command File for zOS

- [Creating Encrypted Command File for z/OS](#)
  - [Command File](#)
    - [Command File Options](#)
  - [JCL](#)
    - [SYSIN Options](#)
  - [Contents of Encrypted File](#)
  - [Components](#)

## Creating Encrypted Command File for z/OS

### Command File

In this example, a Universal Command command file named **MY.CLEAR.COMDFILE** contains the following data:

```
-userid T02JAH1 -pwd thames
```

### Command File Options

The command file options used in this example are:

Option	Description
-userid	User ID or account with which to execute the remote command.
-pwd	Password associated with -userid.

### JCL

The following JCL encrypts the command file allocated to ddname **UNVIN** using AES encryption and an encryption key **MYKEY123**:

```
//UENCRYPT EXEC PGM=UENCRYPT
//STEPLIB DD DISP=SHR,DSN=UNV.SUNVLOAD
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
//UNVIN DD DISP=SHR,MY.CLEAR.COMDFILE
//UNVOUT DD DISP=SHR,MY.ENCRYPT.COMDFILE
//SYSIN DD *
-key MYKEY123 -aes YES
/*
```

The resulting encrypted command file is written to ddname **UNVOUT**.

### SYSIN Options

The SYSIN options used in this example are:

Option	Description
-key	Encryption key used by the encryption algorithm.

**-aes**

Specification for whether or not AES encryption is used.

## Contents of Encrypted File

The figure below illustrates the contents of **MY.ENCRYPT.CMDFILE**.

```
# Universal Encrypt
# Date . . . . . : Thu Nov  3 07:29:03 2011
# User . . . . . : T02JAH1
# Host . . . . . : hosta.acme.com
# Program . . . . : uencrypt 3.2.0 Level 5 Release Build 130
# Encryption . . . : AES 256-bit

1F7DAF62583C813EA874CA168FF626C348F7BF171477D380D9A2FFFED33C539B71B4206EA5021F
D92CDFDD931C3B88B9CD711A4693EFE6B49FAE9431E9C946F7F35C9B4C31335BFB3F97F0686EFF
37068245A6B58CBFE2ADE32997A132C4114AC52CD615B2E7E8672ED0BF9867CA13B1
```

This encrypted command file can now be used by any Workload Automation command on any platform by specifying the encryption key **MYKEY123**.

## Components

Universal Encrypt

## Using Encrypted Command File on zOS

### Using Encrypted Command File on z/OS

For z/OS, the Universal Command Manager `-encryptedfile` option specifies the ddname in the JCL that references the location of the Uencrypted file.

```
//UCM#000 JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*
//          JCLLIB ORDER=#HLQ.UNV.SUNVSAMP
//*
//STEP1    EXEC UCMDPRC
//UENCRYPT DD DISP=SHR,DSN=TEST.UENFILES(TESTPWD)
//COMMANDS DD *
DIR
//SYSIN    DD *
-host          10.252.2.232
-userid        "testid"
-encryptedfile UENCRYPT
-script        COMMANDS
```

### SYSIN Options

The SYSIN options used in this example are:

Option	Description
<code>-host</code>	List of one or more hosts upon which a command can run.
<code>-userid</code>	User ID or account with which to execute the remote command.
<code>-encryptedfile</code>	Encrypted command file.
<code>-script</code>	Local script file to execute on the remote system.

### Components

Universal Command Manager for z/OS

Universal Encrypt

## Creating Encrypted Command File for Windows

- [Creating Encrypted Command File for Windows](#)
  - [Command File](#)
    - [Command File Options](#)
  - [Encryption Command](#)
    - [Command Line Options](#)
  - [Contents of Encrypted File](#)
  - [Components](#)

## Creating Encrypted Command File for Windows

### Command File

In this example, a Universal Command command file named `cmdfile.txt` contains the following data:

```
-userid T02JAH1 -pwd thames
```

### Command File Options

The command file options used in this example are:

Option	Description
<code>-userid</code>	User ID or account with which to execute the remote command.
<code>-pwd</code>	Password associated with <code>-userid</code> .

### Encryption Command

The following command encrypts the command file using AES encryption with an encryption key **MYKEY123**.

```
uencrypt -key MYKEY123 -aes yes < cmdfile.txt > encfile.txt
```

The resulting encrypted command file is written to file `encfile.txt`.

### Command Line Options

The command line options used in this example are:

Option	Description
<code>-key</code>	Encryption key used by the encryption algorithm.
<code>-aes</code>	Specification for whether or not AES encryption is used.

### Contents of Encrypted File

The following figure illustrates the contents of `encfile.txt`.

```
# Universal Encrypt
# Date . . . . . : Thu Nov  3 07:29:03 2011
# User . . . . . : T02JAH1
# Host . . . . . : hosta.acme.com
# Program . . . . : uencrypt 3.2.0 Level 5 Release Build 130
# Encryption . . . : AES 256-bit

1F7DAF62583C813EA874CA168FF626C348F7BF171477D380D9A2FFFED33C539B71B4206EA502
1FD92CDFDD931C3B88B9CD711A4693EFE6B49FAE9431E9C946F7F35C9B4C31335BFB3F97F068
6EFF37068245A6B58CBFE2ADE32997A132C4114AC52CD615B2E7E8672ED0BF9867CA13B1
```

This encrypted command file now can be used by any Workload Automation command, on any operating system, by specifying the encryption key **MYKEY123**.

## Components

[Universal Command Manager for Windows](#)

[Universal Encrypt](#)

## Using Encrypted Command File on Windows

### Using Encrypted Command File on Windows

For Windows, the Universal Command Manager `-encryptedfile` option specifies the location of the Uencrypted file.

```
ucmd -host 10.252.2.232 -userid testid -encryptedfile c:\Universal\Encrypted\enc.txt -cmd "dir"
```

### Command Line Options

The Command options used in this example are:

Option	Description
<code>-host</code>	List of one or more hosts upon which a command can run.
<code>-userid</code>	User ID or account with which to execute the remote command.
<code>-encryptedfile</code>	Encrypted command file.
<code>-cmd</code>	Remote command to execute.

### Components

[Universal Command Manager for Windows](#)

[Universal Encrypt](#)

## Creating Encrypted Command File for UNIX

- [Creating Encrypted Command File for UNIX](#)
  - [Command File](#)
    - [Command File Options](#)
  - [Encryption Command](#)
    - [Command Line Options](#)
  - [Contents of Encrypted File](#)
  - [Components](#)

## Creating Encrypted Command File for UNIX

### Command File

In this example, a Universal Command command file named `cmdfile.txt` contains the following data:

```
-userid T02JAH1 -pwd thames
```

### Command File Options

The command file options used in this example are:

Option	Description
<code>-userid</code>	User ID or account with which to execute the remote command.
<code>-pwd</code>	Password associated with <code>-userid</code> .

### Encryption Command

The following command encrypts the command file using AES encryption with an encryption key **MYKEY123**.

```
uencrypt -key MYKEY123 -aes yes < cmdfile.txt > encfile.txt
```

The resulting encrypted command file is written to file `encfile.txt`.

### Command Line Options

The command line options used in this example are:

Option	Description
<code>-key</code>	Encryption key used by the encryption algorithm.
<code>-aes</code>	Specification for whether or not AES encryption is used.

### Contents of Encrypted File

The following figure illustrates the contents of `encfile.txt`.

```
# Universal Encrypt
# Date . . . . . : Thu Nov  3 07:29:03 2011
# User . . . . . : T02JAH1
# Host . . . . . : hosta.acme.com
# Program . . . . : uencrypt 3.2.0 Level 5 Release Build 130
# Encryption . . . : AES 256-bit

1F7DAF62583C813EA874CA168FF626C348F7BF171477D380D9A2FFFED33C539B71B4206EA502
1FD92CDFDD931C3B88B9CD711A4693EFE6B49FAE9431E9C946F7F35C9B4C31335BFB3F97F068
6EFF37068245A6B58CBFE2ADE32997A132C4114AC52CD615B2E7E8672ED0BF9867CA13B1
```

This encrypted command file now can be used by any Workload Automation command, on any operating system, by specifying the encryption key **MYKEY123**.

## Components

[Universal Command Manager for UNIX](#)

[Universal Encrypt](#)



## Using Encrypted Command File on UNIX

### Using Encrypted Command File on UNIX

For the UNIX, the Universal Command Manager `-encryptedfile` option specifies the location of the Uencrypted file.

```
/opt/universal/bin/ucmd -host 10.252.2.232 -userid testid \  
-encryptedfile /universal/encrypted/encfile.txt -cmd "dir"
```

### Command Line Options

The Command options used in this example are:

Option	Description
<code>-host</code>	List of one or more hosts upon which a command can run.
<code>-userid</code>	User ID or account with which to execute the remote command.
<code>-encryptedfile</code>	Encrypted command file.
<code>-cmd</code>	Remote command to execute.

### Components

[Universal Command Manager for UNIX](#)

[Universal Encrypt](#)

## Creating Encrypted Command File for IBM i

- Creating Encrypted Command File for IBM i
  - Command File
    - Command File Options
  - Encryption Command
    - Command Line Options
  - Contents of Encrypted File
  - Components

## Creating Encrypted Command File for IBM i

### Command File

In this example, a Universal Command command file named **MYLIB/QTXTSRC(TESTLOGIN)** contains the following data:

```
-userid T02JAH1 -pwd tz74gan
```

### Command File Options

The command file options used in this example are:

Option	Description
-userid	User ID or account with which to execute the remote command.
-pwd	Password associated with -userid.

### Encryption Command

The following command encrypts the command file using non-AES encryption with an encryption key **MYKEY123** for default codepage IBM1047.

```
STRUEN INFILE(MYLIB/QTXTSRC) INMBR(TESTLOGIN) OUTFILE(MYLIB/ENCRYPTEDF) OUTMBR(ENCRYPTEDF)
KEY(MYKEY123)
```

The resulting encrypted command file is written to file **ENCRYPTEDF** in **MYLIB** library.

### Command Line Options

The command line options used in this example are:

Option	Description
INFILE	Input file that is to be encrypted.
INMBR	Location of data in the input file that is to be encrypted.
OUTFILE	File to which the encrypted input file is written.

OUTMBR	Location of data in the file to which the encrypted input file is written.
KEY	Encryption key used by the encryption algorithm.

### Contents of Encrypted File

The following figure illustrates the contents of `MYLIB/ENCRYPTEDF (ENCRYPTEDF)`.

```
# Universal Encrypt
# Created on Wed Feb 22 18:43:51 2011
# Created by uencrypt 3.2.0 Level 0

9ACB96416816600CB9D24C9072D80C11768B93CB0E79B944EC37D3495097AD793F97399220C9BB
472DF1E04F5BA8909BCA6C8C72DFD3B706487B1713E6F73F5A0539F17076DEF6D14083EF6E7023
158526E70BE3AF688579805DCAC0CFF1EB6A
```

This encrypted file now can be used as command file input for a Workload Automation command on any platform that uses the encryption key **MYKEY123**.

### Components

Universal Command Manager for IBM i

Universal Encrypt

## Using Encrypted Command File on IBM i

### Using Encrypted Command File on IBM i

For IBM i, the Universal Command Manager `ECMFILE` / `ECMMBR` option specifies the location of the Uencrypted file.

```
STRUCM HOST('10.252.2.232') USERID(testid) ECMFILE(UNIVERSAL/ENCRYPTED) ECMMBR(TETSPWD) CMD('DIR')
```

### Command Line Options

The command line options used in this example are:

Option	Description
HOST	List of one or more hosts upon which a command can run.
USERID	User ID or account with which to execute the remote command.
ECMFILE	Encrypted command file.
ECMMBR	Location of encrypted data in encrypted command file.
CMD	Remote command to execute.

### Components

Universal Command Manager for IBM i

Universal Encrypt

## **Infitran - Configuration Management**

## Configuration Management - Overview

### Configuration Management

Configuration consists of specifying options that control component behavior and resource allocation.

- An example of configurable component 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.

Configuration can be done either by:

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

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 component options, components are – in general – 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 component configuration, there are multiple methods available to configure the components in order to meet your needs.

## Configuration Methods

## Configuration Methods - Overview

- Configuration Methods
  - Universal Broker / Servers Configuration Method
  - z/OS Platform

### Configuration Methods

All components 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 the specific Workload Automation component, and the operating system on which it is being run, component configuration is performed by one or more methods.

These configuration methods, in their order of precedence, are:

1. [Command Line](#)
2. [Command File](#)
3. [Environment Variables](#)
4. [Configuration File](#)

The command line, command file, and environment variables methods let you set configuration options and preferences for a single execution of a component.

The configuration file method lets you set default configuration options and preferences for all executions of a component.

This order of precedence means that an 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 in a configuration file.



#### Note

For security reasons, not all options can be overridden.

### Universal Broker / Servers Configuration Method

Universal Broker, and all Workload Automation servers, are configurable only by modifying their configuration files (see [Configuration File](#)). They are not configurable via command line, command file, or environmental variables.

### z/OS Platform

On the z/OS platform, configuration can utilize z/OS system symbols as part of the configuration value. Each system symbol is resolved when the value is first read by a component.

z/OS System symbols may be used in some of the configuration methods as follows:

- Command line or command file options prefixed with a plus (+) character instead of a dash (-) result in system symbols in the option value being resolved.
- System symbols are not supported in environment variables.
- System symbols are always resolved in configuration file values.

System symbols start with the ampersand character (&) and end with a period (.). For example, the **&SYSNAME.** symbol specified in the Universal Broker **UNIX\_DB\_DATA\_SET** option is "**UNV.&SYSNAME..UNVDB**". The variable "**&SYSNAME.**" will be replaced with the symbol value.

The z/OS system symbols that are defined on z/OS can be displayed with the MVS system command **DISPLAY SYMBOLS**.



## Configuration Methods - Command Line

- [Command Line](#)
  - [z/OS](#)
  - [UNIX, Windows, and HP NonStop](#)
  - [IBM i](#)

### Command Line

Command line options affect one instance of a program execution. Each time that you execute a program, command line options let you tailor the behavior of the program to meet the specific needs for that execution.

Command line options are the highest in order of precedence of all the configuration methods (see [Configuration Methods](#)). They override the options specified using all other configuration methods, except where indicated.

Each command line options consist of:

- Parameter (name of the option)
- Value (pre-defined or user-defined value of the option)

The command line syntax depends, in part, on the operating system, as noted below.

A value may or may not be case-sensitive, depending on what it is specifying. For example, if a value is either **yes** or **no**, it is not case-sensitive. It could be specified as **YES**, **Yes**, or **yes**. However, if a value specifies a directory name or file name, it would be case-sensitive if the operating system's file system is case-sensitive.

If an option is specified more than once on the command line, the last instance of the option specified is used.

#### ***z/OS***

z/OS command line options are specified in the JCL EXEC statement PARM keyword or on the SYSIN ddname. The PARM keyword is used to pass command line options to the program being executed with the EXEC statement.

Command line options are prefixed with a dash (-) character or a plus (+) character. The plus character indicates that system symbols found in the value are resolved to their defined value before the value is processed by the Workload Automation component. For many options, there are two different forms in which they can be specified:

- Short form: one case-sensitive character
- Long form: two or more case-insensitive characters

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

Example command line options specified in the PARM value follow:

```
Short form:
PARM='-l INFO -G yes'

Long form:
PARM='-LEVEL INFO -LOGIN YES'
```

As noted above, z/OS command line options also can be specified on the SYSIN ddname. This is the easiest and least restrictive place to specify options, since the PARM values are limited in length. The options specified in the SYSIN ddname have the same syntax. Options can be specified on one line or multiple lines. The data set or inline data allocated to the SYSIN ddname cannot have line numbers in the last 8 columns (that is, all columns of the records are used as input).

#### ***UNIX, Windows, and HP NonStop***

UNIX, Windows, and HP NonStop command line options are prefixed with a dash (-) character, and alternatively on Windows, the slash (/) character.

For many options, there are two different forms in which they can be specified:

- Short form: one case-sensitive character.
- Long form: two or more case insensitive characters.

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

Example command line options follow:

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

### ***IBM i***

IBM i command line options use the native conventions for Command Language (CL) commands. The option name is specified as a CL parameter with its value enclosed in parentheses.

Example command line options follow:

```
Command line options:  
MSGLEVEL(INFO) COMPRESS(\*YES)
```

All Workload Automation 5 components provide IBM i-style command panels. The panels are accessed by entering the command name on the command line and pressing the F4 (PROMPT) key.

## Configuration Methods - Command File

### Command File

The command file contains command line options specified in a file. The command file enables you to save common command line options in permanent storage and reference them as needed.

The command file is the second to highest in the precedence order, after command line options (see [Configuration Methods](#)).

Individual command line options can be specified on one or multiple lines. Blank lines are ignored. Lines starting with the hash ( # ) character are ignored and can be used for comments.

The command file can be encrypted if it is necessary to secure the contents.



#### Note

If the contents of the file contain sensitive material, the operating system's native file and user security facilities should be used in addition to the file encryption provided by Workload Automation.

In order to use a command file, either of the following is used:

- `COMMAND_FILE_PLAIN` option is used to specify the command file name.
- `COMMAND_FILE_ENCRYPTED` option is used to specify the encrypted command file name.

## Configuration Methods - Environment Variables

- [Environment Variables](#)

### 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, environment variables allow you to tailor the behavior of the program to meet the specific needs for that execution.

Environment variables are the third to highest in the precedence order, after command file options (see [Configuration Methods](#)).

Each operating system has its own unique method of setting environment variables.

All environment variables used by Workload Automation are upper case and are prefixed with a product identifier consisting of three or four characters. The product sections specify the value of the environment variables. Values are case-sensitive.

<p><b>z/OS</b></p>	<p>Environment variables in z/OS are specified in the JCL EXEC statement PARM keyword. 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.                  Example of setting an environment variable (set option UCMDLEVEL to a value of INFO):</p> <pre style="border: 1px solid #ccc; padding: 10px; margin: 10px 0;">                 PARM= ' ENVAR ( "UCMDLEVEL=INFO" ) / '             </pre>
<p><b>UNIX</b></p>	<p>Environment variables in UNIX are defined as part of the shell environment. As such, shell commands are used to set environment variables. The environment variable must be exported to be used by a called program.                  Example of setting an environment variable (set option UCMDLEVEL to a value of INFO in a bourne, bash, or korn shell):</p> <pre style="border: 1px solid #ccc; padding: 10px; margin: 10px 0;">                 UCMDLEVEL=INFO                 export UCMDLEVEL             </pre>
<p><b>Windows</b></p>	<p>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 UCMDLEVEL to a value of INFO):</p> <pre style="border: 1px solid #ccc; padding: 10px; margin: 10px 0;">                 SET UCMDLEVEL=INFO             </pre>
<p><b>IBM i</b></p>	<p>Environment variables in IBM i are defined with Command Language (CL) commands for the current job environment.                  Example of setting an environment variable (set option UCMDLEVEL to a value of INFO):</p> <pre style="border: 1px solid #ccc; padding: 10px; margin: 10px 0;">                 ADDENVVAR ENVVAR(UCMDLEVEL) VALUE(INFO)             </pre>
<p><b>HP NonStop</b></p>	<p>Environment variables in HP NonStop are defined with HP NonStop Advanced Command Language (TACL) commands for the current job environment.                  Example of setting an environment variable (set option UCMDLEVEL to a value of INFO):</p> <pre style="border: 1px solid #ccc; padding: 10px; margin: 10px 0;">                 PARAM UCMDLEVEL INFO             </pre>



## Configuration Methods - Configuration File

- [Configuration File](#)
- [Configuration File Syntax](#)

### Configuration File

Configuration files are used to specify system-wide configuration values. This method is last in the order of precedence; that is, configuration file option values can be overridden by every other method of configuration (see [Configuration Methods](#)).

(For most Workload Automation components, some options can be specified only in a configuration file, while other options can be overridden by individual command executions. The Workload Automation Reference Guide for each component identifies these options.)

The configuration files for all Workload Automation components on a system are maintained by the local Universal Broker. Universal Broker serves the configuration data to the other Workload Automation components. The components do not read the configuration files themselves (except for Universal Enterprise Controller, which does read its own configuration files).

When a component starts, it first registers with the locally running Universal Broker. As part of the registration process, the Broker returns the component's configuration data to the component.

Universal Broker reads the configuration files when it first starts or when it receives a configuration refresh request 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 configuration refresh request (see [Configuration Refresh](#)).

Universal Broker can operate in managed or unmanaged mode:

- In unmanaged mode, the configuration information for the various Workload Automation components can be modified either:
  - Locally (either by editing the configuration files or, on Windows systems, via the [Universal Configuration Manager](#)).
  - Remotely, via the Universal Enterprise Controller [I-Management Console](#) application.
- In managed mode, the configuration information for the various Workload Automation components is "locked down" and can be modified or viewed only via the I-Management Console.

(For information on unmanaged and managed modes, see [Remote Configuration](#)).

<b>z/OS</b>	<p>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.</p> <p>All configuration files are installed in the <b>UNVCONF</b> library. See <a href="#">Configuration File Syntax</a>, below, for the configuration file syntax.</p>
<b>UNIX</b>	<p>Configuration files are regular text files on UNIX. The files can be edited with a text editor. See <a href="#">Configuration File Syntax</a>, below, for the configuration file syntax.</p> <p>Universal Broker searches for the configuration files in a fixed list of directories. The Broker will use the first configuration file that it finds in its search. The directories are listed below in the order they are searched.</p> <ul style="list-style-type: none"> <li>• /etc/opt/universal</li> <li>• /etc/universal (installation default)</li> <li>• /etc/stonebranch (obsolete as of version 2.2.0)</li> <li>• /etc</li> <li>• /usr/etc/universal</li> <li>• /usr/etc/stonebranch (obsolete as of version 2.2.0)</li> <li>• /usr/etc</li> </ul>
<b>Windows</b>	<p>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.</p> <p>The <a href="#">Universal Configuration Manager</a> 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.</p>
<b>IBM i</b>	<p>The configuration files on IBM i are stored in a source physical file named UNVCONF in the UNVPRD510 library. The files can be edited with a text editor. See <a href="#">Configuration File Syntax</a>, below, for the configuration file syntax.</p>
<b>HP NonStop</b>	<p>The configuration files on HP NonStop are stored as EDIT files, file code 101, within the <b>\$SYSTEM.UNVCONF</b> subvolume. The files can be edited with the EDIT editor. See <a href="#">Configuration File Syntax</a>, below, for the configuration file syntax.</p>

### 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.

## Configuration Options

### Infitran Configuration Options

The following configuration options are available for Infitran components:

Universal Automation Center Agent Configuration Options

Universal Broker Configuration Options

Universal Command Manager Configuration Options

Universal Command Server Configuration Options

Universal Command Agent for SOA Configuration Options

Universal Connector Configuration Options

Universal Data Mover Manager Configuration Options

Universal Data Mover Server Configuration Options

Universal Enterprise Controller Configuration Options

UECLoad Configuration Options

Universal Event Monitor Manager Configuration Options

Universal Event Monitor Server Configuration Options

UEMLoad Configuration Options

Universal Certificate Configuration Options

Universal Control Manager Configuration Options

Universal Control Server Configuration Options

Universal Copy Configuration Options

Universal Database Dump Configuration Options

Universal Database Load Configuration Options

Universal Display Log File Configuration Options

Universal Encrypt Configuration Options

Universal Event Log Dump Configuration Options

Universal Message Translator Configuration Options

Universal Products Install Merge Configuration Options

Universal Query Configuration Options

Universal Spool List Configuration Options

Universal Spool Remove Configuration Options

Universal Submit Job Configuration Options

Universal Write-to-Operator Configuration Options



## Remote Configuration

- Remote Configuration
- Unmanaged Mode
- Managed Mode
  - Selecting Managed Mode
- Universal Broker Start-up
  - Unmanaged Mode
  - Managed Mode

### Remote Configuration

Workload Automation components can be configured remotely by Universal Enterprise Controller using the I-Management Console client application, and can be "locked down" so that they *only* can be remotely configured.

I-Management Console instructs the Universal Broker of a remote Agent to modify the configurations of all Workload Automation components managed by that Universal Broker.

Universal Broker supports remote configuration in either of two modes:

1. Unmanaged Mode
2. Managed Mode

### Unmanaged Mode

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

- Locally, by editing configuration files.
- Remotely, via I-Management Console.

The system administrator for the machine on which an Agent resides can use any text editor to modify the configuration files of the various local Workload Automation components.

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

If I-Management Console 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 I-Management Console sends modification to the configuration of any other Workload Automation 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 I-Management Console for a component other than Universal Broker, the component may not run successfully until the configuration has been corrected.

### Managed Mode

When a Universal Broker is operating in managed mode, the configuration information for all Workload Automation 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 I-Management Console.

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 I-Management Console – 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 Workload Automation servers – no longer support the command line and environmental variables methods of specifying configuration options.

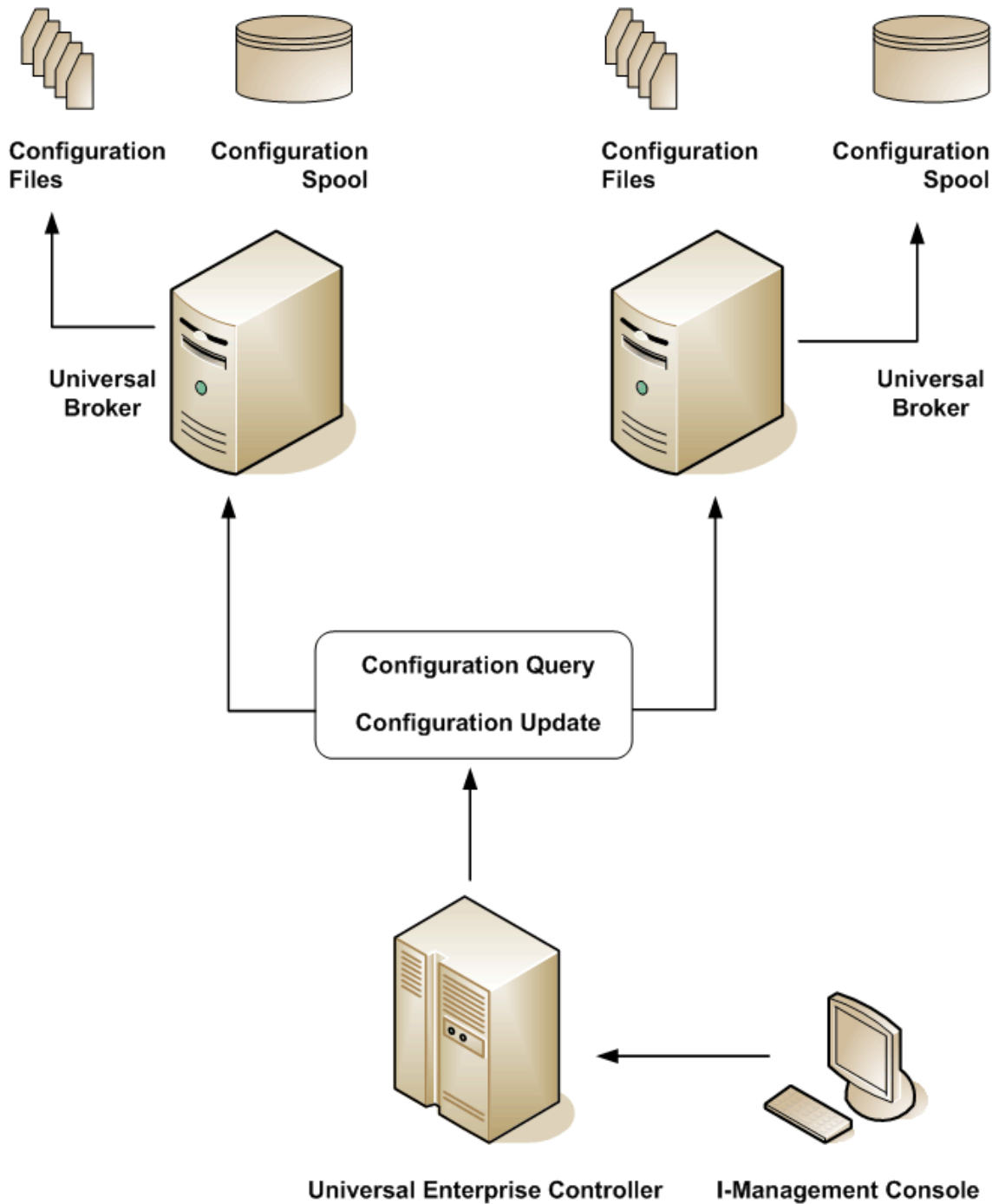
## Selecting Managed Mode

The managed mode of operations for Universal Broker is selected via the [I-Administrator](#) client application (see [Adding an Agent](#)).

The following figure illustrates remote configuration for one Agent in managed mode and one Agent in unmanaged mode.

## Agent – Unmanaged Mode

## Agent – Managed Mode



### Universal Broker Start-up

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

### Unmanaged Mode

At Universal Broker start-up in unmanaged mode, Universal Broker reads the configuration files of all Workload Automation 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 configuration 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 Workload Automation components. Universal Broker updates its memory from the configuration spool file and, automatically, after changes are made via I-Management Console.

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.

## Universal Configuration Manager

## Universal Configuration Manager - Usage

- Overview
- Availability
  - Windows Vista, Windows 7
- Accessing the Universal Configuration Manager
- Navigating through Universal Configuration Manager
- Modifying / Entering Data
  - Rules for Modifying / Entering Data
- Saving Data
- Accessing Help Information

### Overview

The Universal Configuration Manager is a Workload Automation 5 graphical user interface application that enables you to configure all of the Workload Automation 5 components 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.

### Availability

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

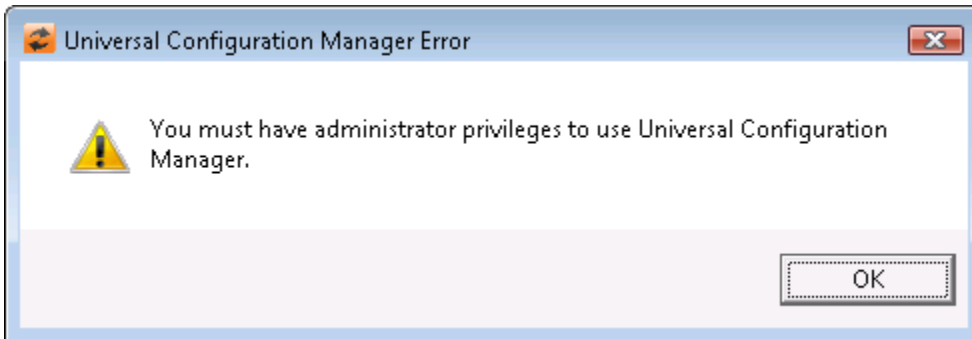
It is available to all user accounts in the Windows Administrator group.

### Windows Vista, Windows 7

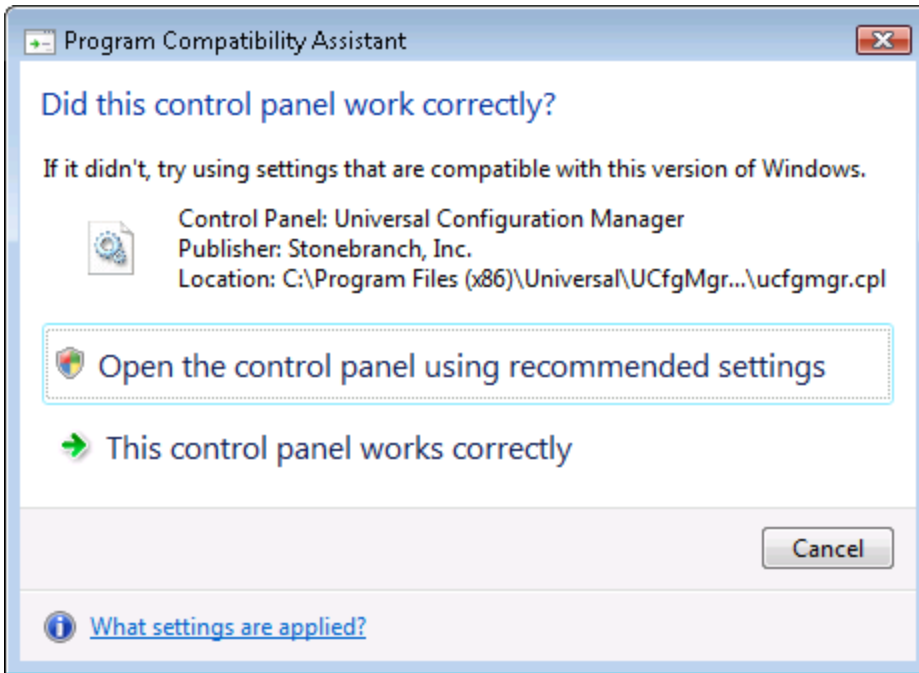
When opening the Universal Configuration Manager for the first time on Windows Vista / Windows 7, two new operating system features, the Program Compatibility Assistant (PCA) and User Account Control (UAC), may affect its behavior.

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

1. Universal Configuration Manager may issue the following error.



2. Click **OK** to dismiss the error message.  
The Windows Vista / Windows 7 Program Compatibility Assistant (PCA) displays the following dialog:



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 / Windows 7 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.

## Accessing the Universal Configuration Manager

To access the Universal Configuration Manager:

<b>Step 1</b>	Click the <b>Start</b> icon at the lower left corner of your Windows operating system screen to display the Start menu.
<b>Step 2</b>	Click (Settings/) Control Panel on the Start menu to display the Control Panel screen.
<b>Step 3</b>	Select the Universal Configuration Manager icon to display the Universal Configuration Manager screen (see the following figure).



### Windows XP, Windows Vista, Windows 7, Windows Server 2008 / 2008 R2

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, Windows 7, and Windows Server 2008 / 2008 R2 place the icon within the **Additional Options** category.

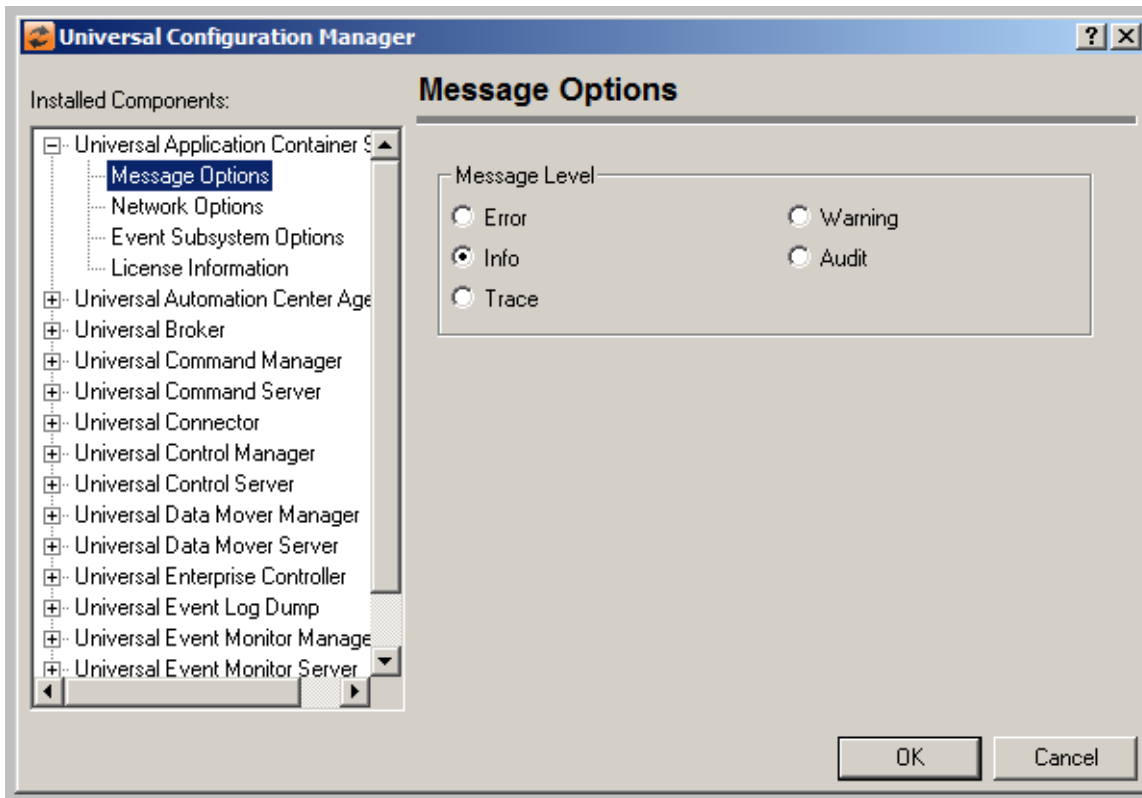
If you have trouble locating the Universal Configuration Manager icon, simply switch to the Classic View to display all Control Panel icons at the same time.



### 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.



Each Universal Configuration Manager screen contains two sections:

1. Left side of the screen displays the Installed Components tree, which lists:
  - Workload Automation 5 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.)

## 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 the previous figure, 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.

## 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 [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.

**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 file, 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.

**Accessing Help Information**

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

To access Help:

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

**Windows Vista, Windows 7, Windows Server 2008 / 2008 R2**

The Universal Configuration Manager's context-sensitive help is a WinHelp file, which Windows Vista, Windows 7, and Windows Server 2008 / 2008 R2 do 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.

## Universal Configuration Manager - Installed Components

- Universal Data Mover Installed Components
  - Universal Data Mover Manager
  - Universal Data Mover Server
- Universal Event Monitor Installed Components
  - Universal Event Monitor Manager
  - Universal Event Monitor Server
- Universal Enterprise Controller Component
- Universal Broker Installed Component
- Universal Automation Center Agent Installed Component
- Workload Automation 5 Utilities Installed Components
  - Universal Control Manager
  - Universal Control Server
  - Universal Event Log Dump
  - Universal Query

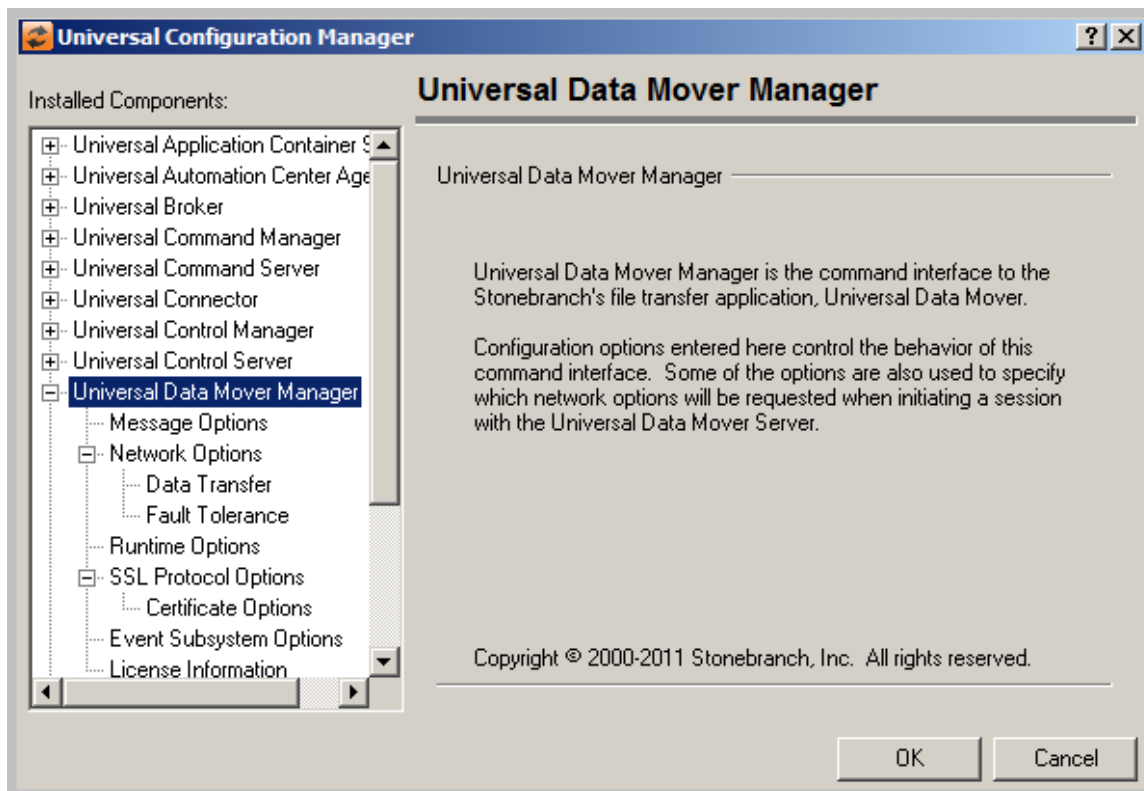
### Universal Data Mover Installed Components

#### Universal Data Mover Manager

The following figure illustrates the Universal Configuration Manager screen for the Universal Data Mover Manager.

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

The text describes the selected component, Universal Data Mover Manager.

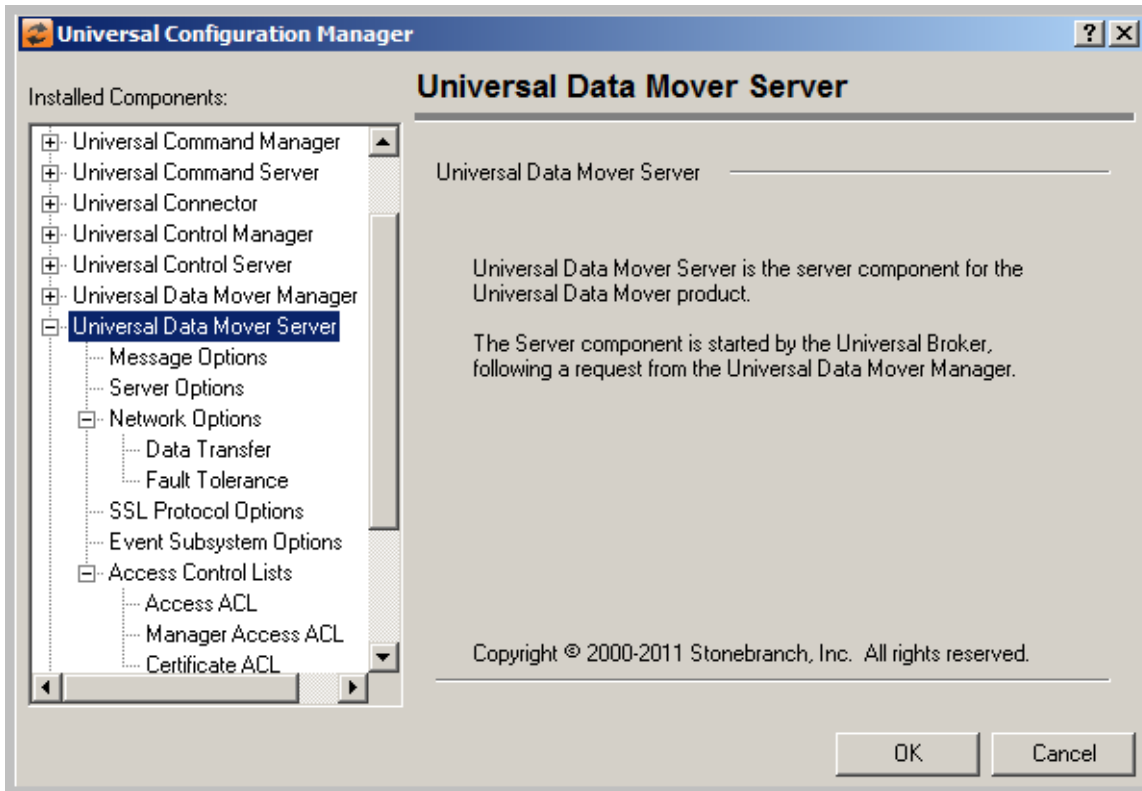


#### Universal Data Mover Server

The following figure illustrates the Universal Configuration Manager screen for the Universal Data Mover Server.

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

The text describes the selected component, Universal Data Mover Server.



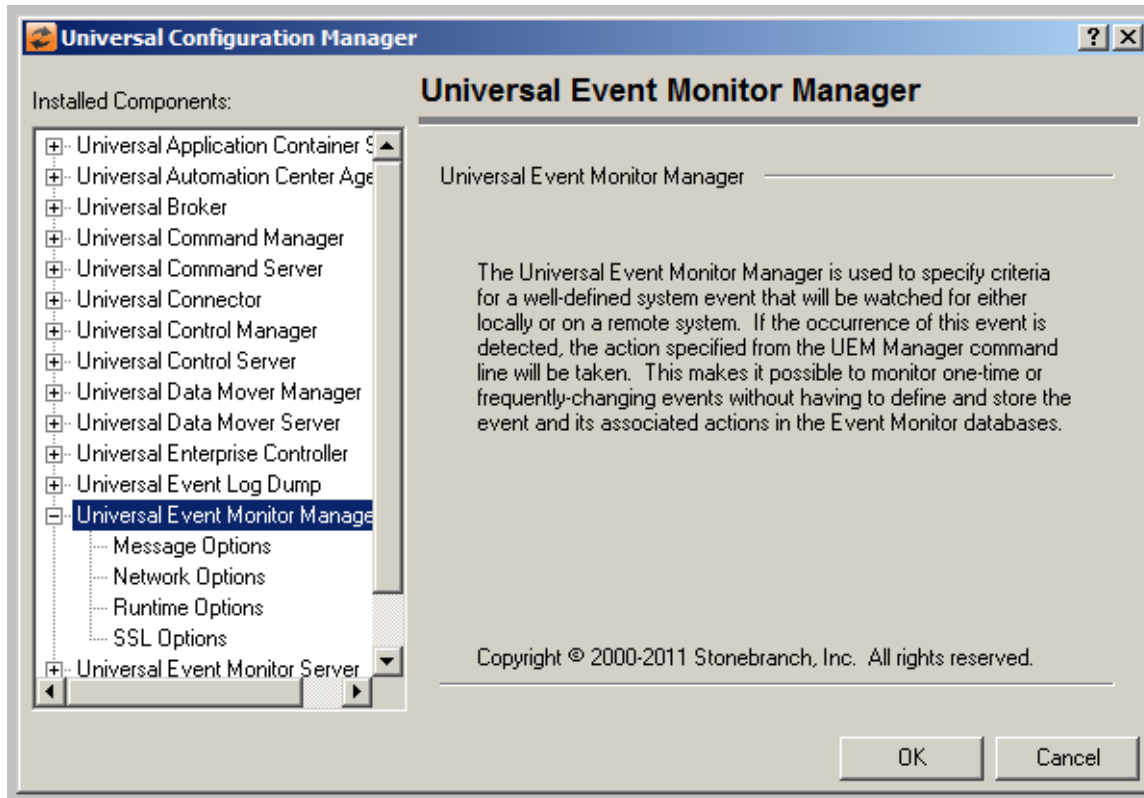
## Universal Event Monitor Installed Components

### Universal Event Monitor Manager

The following figure 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.

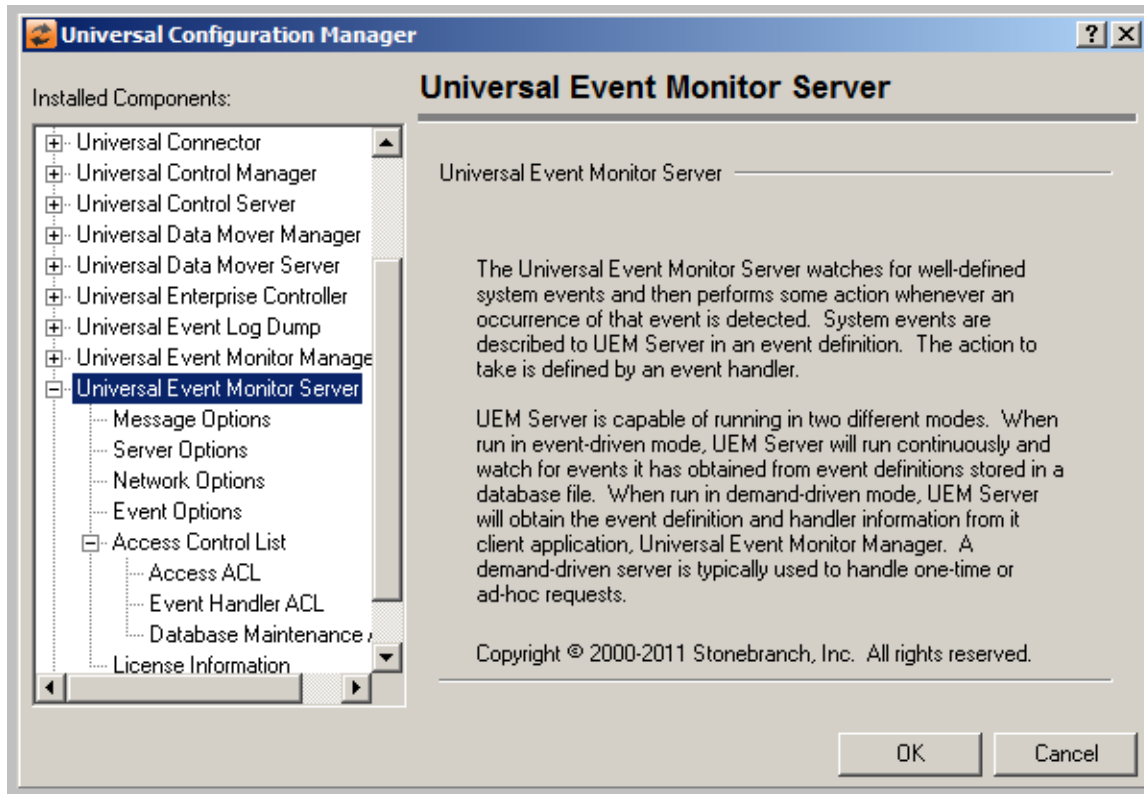


### Universal Event Monitor Server

The following figure 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.

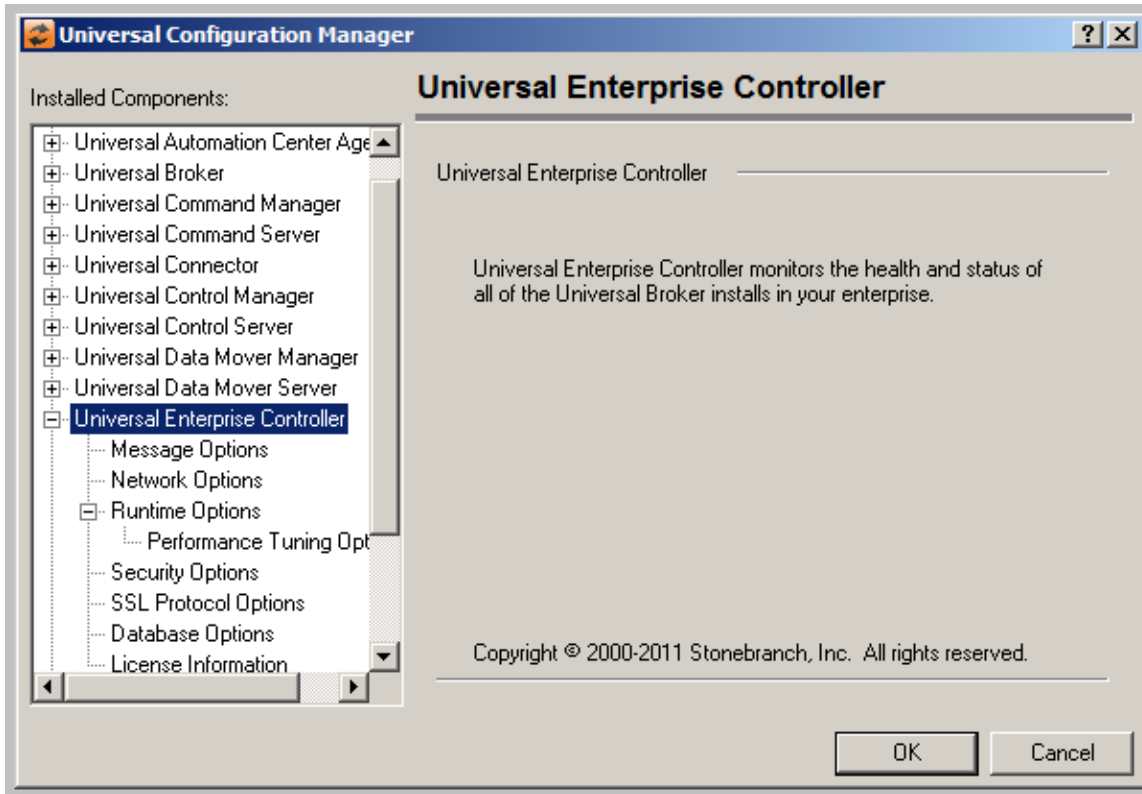


### Universal Enterprise Controller Component

The following figure illustrates the Universal Configuration Manager screen for the Universal Enterprise Controller.

The Installed Components list identifies all of the UEC property pages.

The text describes the selected component, Universal Enterprise Controller.

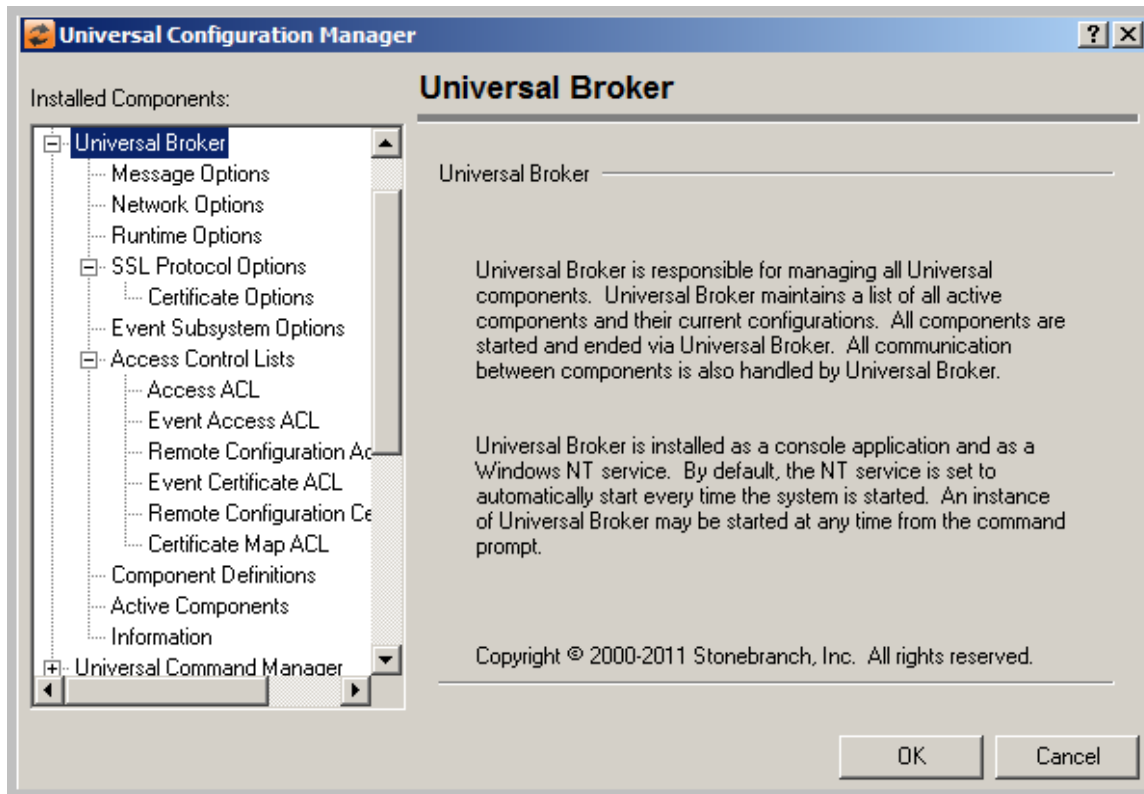


### Universal Broker Installed Component

The following figure illustrates the Universal Configuration Manager screen for the Universal Broker.

The Installed Components list identifies all of the Universal Broker property pages.

The text describes the selected component, Universal Broker.

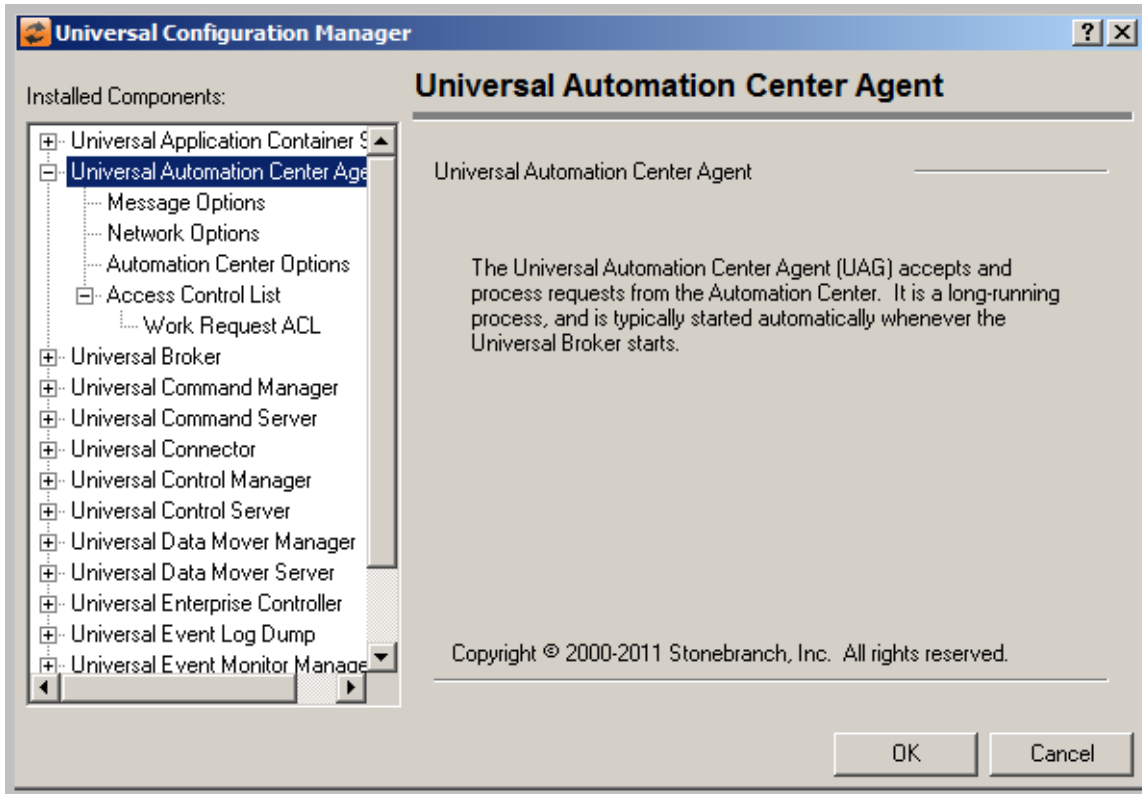


### Universal Automation Center Agent Installed Component

The following figure illustrates the Universal Configuration Manager screen for the Universal Automation Center Agent.

The Installed Components list identifies all of the Universal Automation Center Agent property pages.

The text describes the selected component, Universal Automation Center Agent.



## Workload Automation 5 Utilities Installed Components

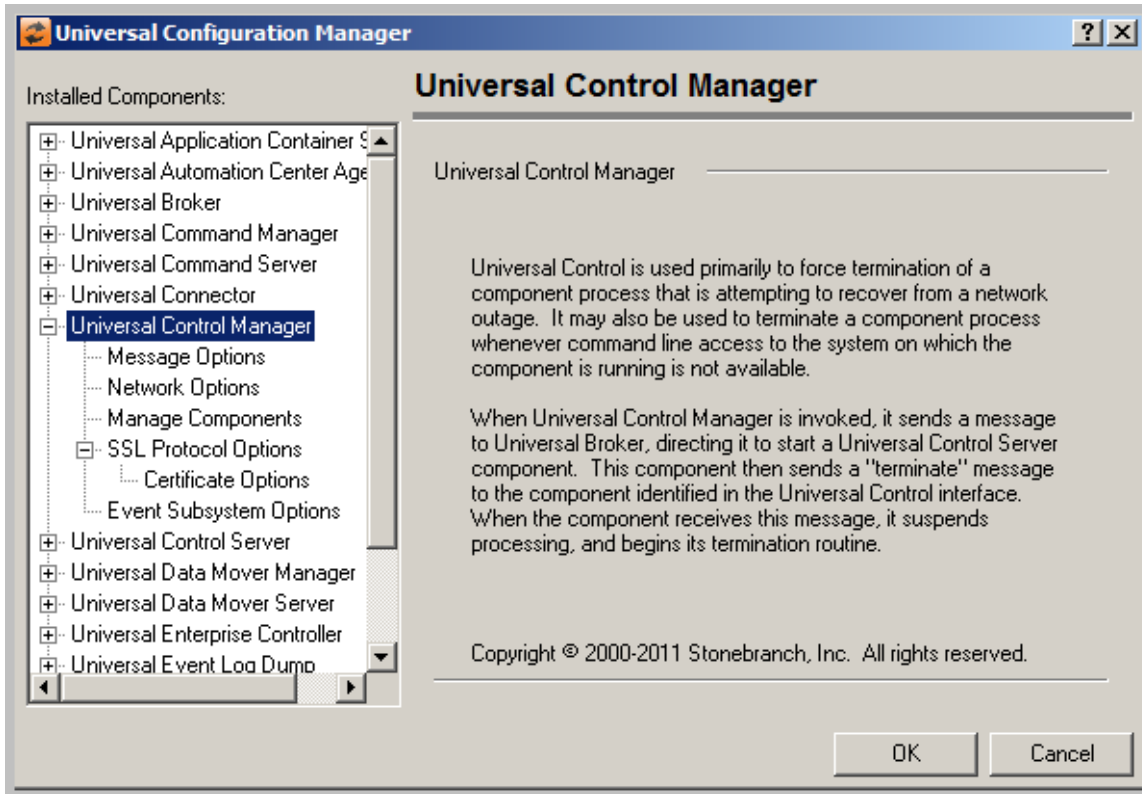
### Universal Control Manager

The following figure illustrates the Universal Configuration Manager screen for the Universal Control Manager.

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

The text describes the selected component, Universal Control Manager.



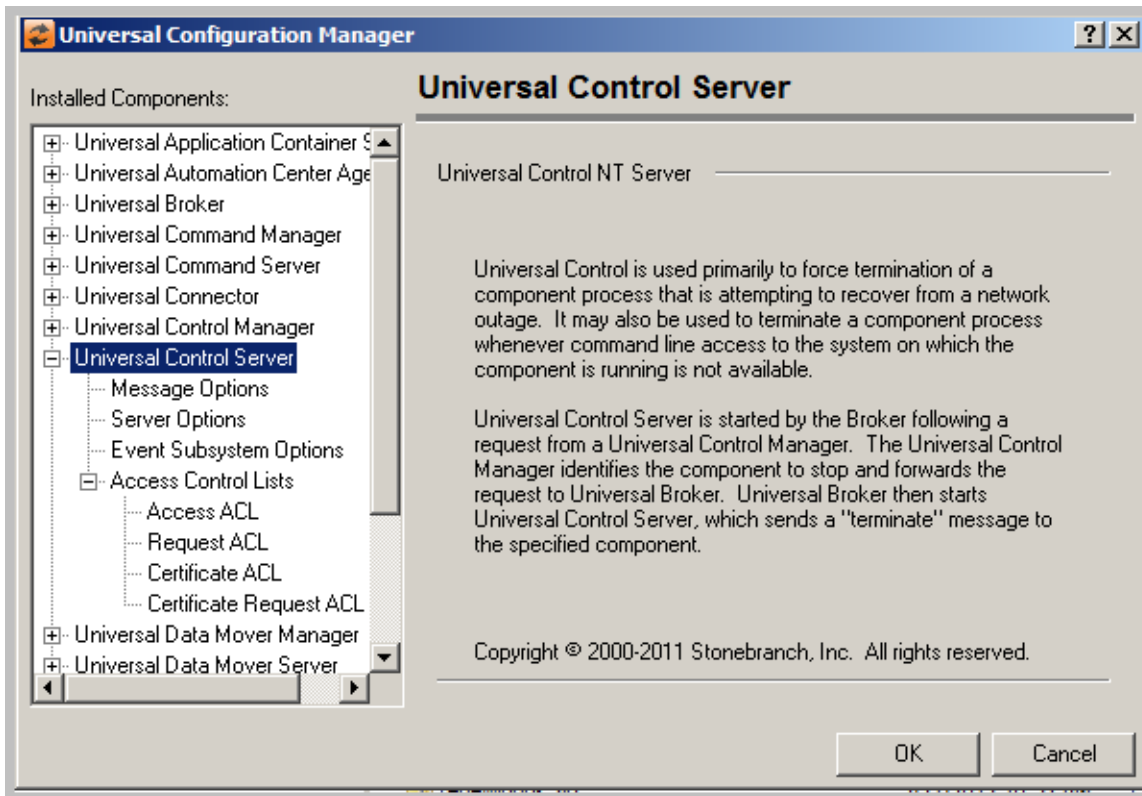


### Universal Control Server

The following figure illustrates the Universal Configuration Manager screen for the Universal Control Server.

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

The text describes the selected component, Universal Control Server.

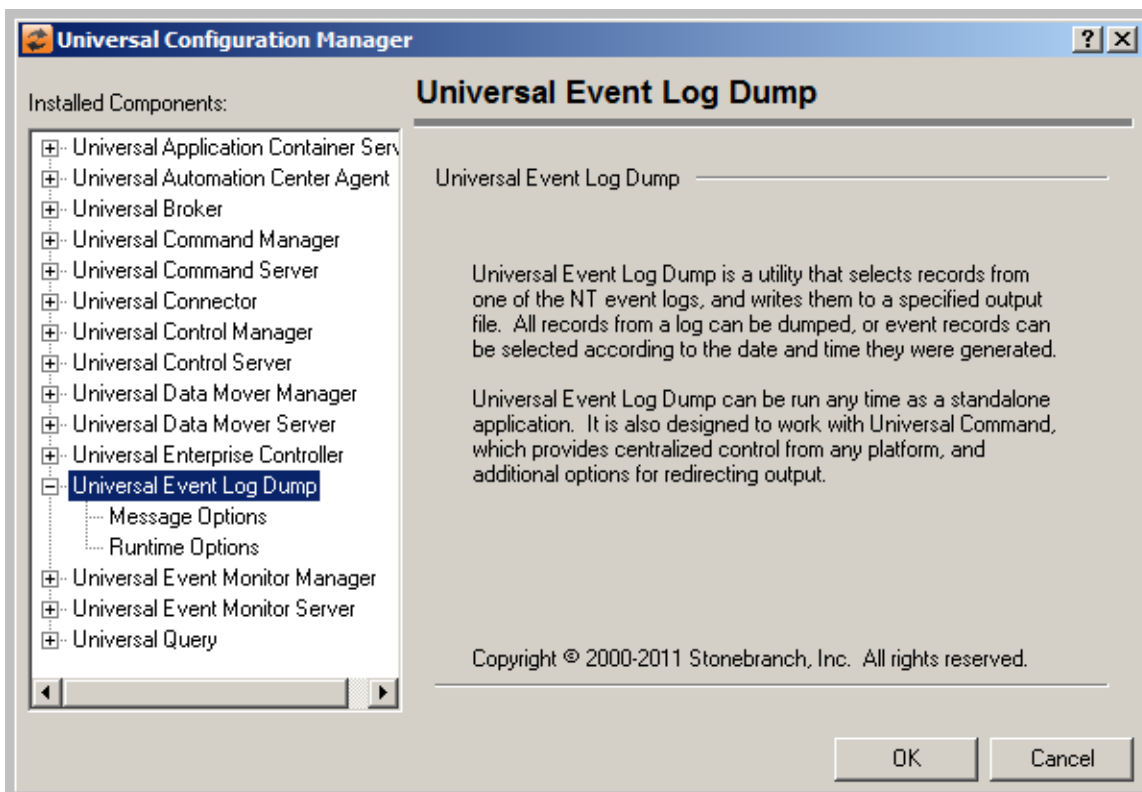


### Universal Event Log Dump

The following figure illustrates the Universal Configuration Manager screen for the Universal Event Log Dump utility.

The Installed Components list identifies all of the Universal Event Log Dump property pages.

The text describes the selected component, Universal Event Log Dump.

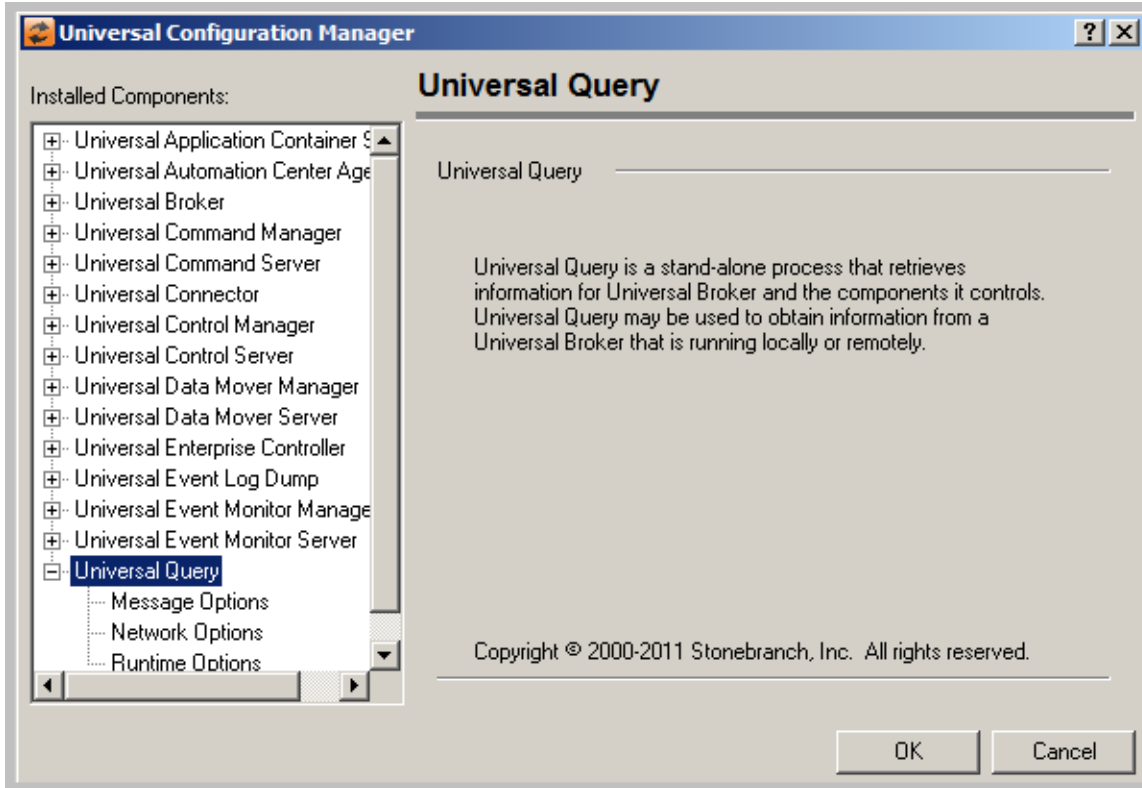


## Universal Query

The following figure illustrates the Universal Configuration Manager screen for the Universal Query utility.

The Installed Components list identifies all of the Universal Query property pages.

The text describes the selected component, Universal Query.



## Configuration Refresh

- [Overview](#)
- [Configuration Refresh via Universal Control](#)
  - [Configuration Refresh for Universal Event Monitor Server](#)
- [Configuration Refresh via Universal Configuration Manager](#)
- [Universal Broker Configuration Options Refresh](#)

### Overview

Universal Broker maintains the configuration files for all Workload Automation 5 components that it manages. The components do not read their configuration files themselves (except for Universal Enterprise Controller, which does read its own configuration file).

When a component starts, it first registers with its local Universal Broker. As part of the registration process, Universal Broker returns the configuration data to the component.

Universal Broker reads the configuration files at initial start-up and, thereafter, whenever it is refreshed; that is, when either of the following occurs:

- Universal Broker is recycled (stopped and restarted).
- Universal Broker is refreshed by Universal Control.
- Universal Broker is refreshed by Universal Enterprise Controller (via I-Management Console).
- Universal Broker is refreshed by Universal Configuration Manager (**Windows only**).

After a configuration file has been modified, the Universal Broker must be refreshed in order for the modified values to take effect. Refreshing a Universal Broker directs it to read its configuration data and update its current configuration settings.

### Configuration Refresh via Universal Control

Universal Control refreshes the Universal Broker by issuing a configuration refresh request via its `REFRESH_CMD` configuration option.

Universal Control directs Universal Broker to refresh the configuration data of all components, including itself, or a single component. (Currently, the only individual component that can be refreshed this way is the Universal Event Monitor Server, `uems`.)

### Configuration Refresh for Universal Event Monitor Server

Because an event-driven Universal Event Monitor (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 [Configuration File](#)), active event-driven UEM Servers must be notified that a change has taken place. This is done via Universal Control, using the Universal Control Manager `REFRESH_CMD` option, along with a component type value that identifies the component to refresh (see [Refreshing via Universal Control - Examples Overview](#)).



#### 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 [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.

### Configuration Refresh via Universal Configuration Manager

When any of the options that can be refreshed are updated using the [Universal Configuration Manager](#), a configuration refresh request is sent to Universal Broker, and its configuration is refreshed automatically.

The configuration refresh request directs Universal Broker to take the following actions:

<b>Step 1</b>	Read its configuration file. Universal Broker refreshes its configuration options.
<b>Step 2</b>	Read all component definitions found in the component definition directory. The Broker replaces all component definitions with the newly read component definitions. New component definitions are added and deleted component definitions are removed.
<b>Step 3</b>	Read the Universal Access Control List configuration entries from the registry. The Broker replaces its UACL entries with the newly read entries.

## Universal Broker Configuration Options Refresh

As with all Workload Automation 5 components, all Universal Broker options can be modified by editing the configuration file directly.

However, unlike other components, not all Universal Broker options can be modified via I-Management Console. In I-Management Console, these Universal Broker options are read-only.

Some Universal Broker options can be modified only by editing the Universal Broker configuration file, **ubroker.conf**. For these modifications to take effect, Universal Broker must be recycled.

All other Universal Broker options can be modified either:

- By editing **ubroker.conf**.
- Via I-Management Console.
- Via the [Universal Configuration Manager](#).

Depending on the option, for a modification to take effect:

- Universal Broker must be recycled.
- Universal Broker must be refreshed by issuing a Universal Control configuration refresh request (via the [REFRESH\\_CMD](#) configuration option), if the modifications are made in **ubroker.conf**.
- Universal Broker is refreshed automatically, if the modifications are made via I-Management Console or the [Universal Configuration Manager](#).

For a list of the Universal Broker configuration options in each category, see [Universal Broker Configuration Options Refresh](#).

## Refreshing via Universal Control Examples

### Refreshing via Universal Control Examples

- Refreshing Universal Broker from z/OS
- Refreshing a Component from z/OS
- Refreshing Universal Broker from Windows
- Refreshing a Component from Windows
- Refreshing Universal Broker from UNIX
- Refreshing a Component from UNIX
- Refreshing Universal Broker from IBM i
- Refreshing a Component from IBM i

***These examples illustrate how to use Universal Control to refresh configuration data of all components, including itself, or a single component. (Currently, the only individual component that can be refreshed is the Universal Event Monitor Server.)***



#### **Note**

The IBM i examples reference the IBM i commands by their untagged names. If you are using commands with tagged names to run **Universal Control**, substitute the tagged names for these untagged names. (For information on tagged names, see [UCHGRLS \(Change Release Tag\) Program](#).)

## Refreshing Universal Broker from z/OS

- Refreshing Universal Broker from z/OS
  - SYSIN Options
  - Universal Broker Actions
  - Components

### Refreshing Universal Broker from z/OS

This example refreshes Universal Broker on z/OS.

```
//jobname JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*****
//* (c) Copyright 2001-2008, Stonebranch, Inc. All rights reserved.
//*
//* Stonebranch, Inc.
//* Universal Control
//*
//* Description
//* -----
//* This sample demonstrates the use of the UCTL program to refresh
//* a running component on host dallas.
//*
//* Make the following modifications as required by your local
//* environment:
//*
//* - Modify the JOB statement as appropriate.
//* - Change all '#HLQ' to the high-level qualifier of the
//*   Universal Command data sets.
//* - If not already done, modify the JCL procedure UCTLPRC
//*   as required by your local environment.
//* *****
//*
//*          JCLLIB ORDER=#HLQ.UNV.SUNVSAMP
//*
//STEP1 EXEC UCTLPRC
//SYSIN DD *
-refresh -host dallas
/*
```

This example refreshes the Universal Broker configuration on host **dallas**.

#### SYSIN Options

The SYSIN options used in this example are:

Option	Description
<code>-refresh</code>	Instruction to refresh Universal Broker on the remote system.
<code>-host</code>	Directs the command to a computer with a host name of <b>dallas</b> .

#### Universal Broker Actions

The refresh request directs the Broker to take the following actions:

<b>Step 1</b>	Read its configuration file. The Broker refreshes configuration options.
<b>Step 2</b>	Read all component definitions found in ddname <b>UNVCONF</b> . The Broker replaces all component definitions with the newly read component definitions. New component definitions are added and deleted component definitions are removed.

<b>Step 3</b>	Read the Universal Access Control List configuration file allocated to ddname <b>UNVACL</b> . The Broker replaces its UACL entries with the newly read entries.
---------------	---

## Components

Universal Control



## Refreshing a Component from zOS

- Refreshing a Component from z/OS
  - SYSIN Options
  - Components

### Refreshing a Component from z/OS

This example refreshes a component on a remote system.

```
//jobname JOB CLASS=A,MSGCLASS=X
//STEP1 EXEC UCTLPRC
//SYSIN DD *
-refresh uems -cmdid "UEM-dallas" -host dallas -userid joe -pwd akkSdiq
/*
```

### SYSIN Options

The SYSIN options used in this example are:

Option	Description
-refresh	Type of component to refresh on the remote system.
-cmdid	Assigns a command identifier of "UEM-dallas" to the started component.
-host	Directs the command to a computer with a host name of <b>dallas</b> .
-userid	Remote user ID with which to execute the Universal Control Server process.
-pwd	Password for the user ID.

### Components

[Universal Control](#)

## Refreshing Universal Broker from Windows

- [Refreshing Universal Broker via Universal Control from Windows](#)
  - [Command Line Options](#)
  - [Universal Broker Actions](#)
  - [Components](#)

### Refreshing Universal Broker via Universal Control from Windows

This example refreshes Universal Broker on a remote system.

```
uctl -refresh -host dallas -userid joe -pwd akkSdiq
```

#### Command Line Options

The command line options used in this example are:

Option	Description
<code>-refresh</code>	Instruction to refresh Universal Broker on the remote system.
<code>-host</code>	Directs the command to a computer with a host name of <b>dallas</b> .
<code>-userid</code>	Remote user ID with which to execute the Universal Control Server process.
<code>-pwd</code>	Password for the user ID.

#### Universal Broker Actions

This refresh request directs Universal Broker to take the following actions:

<b>Step 1</b>	Read its configuration file. Universal Broker refreshes its configuration options.
<b>Step 2</b>	Read all component definitions found in the component definition directory. Universal Broker replaces all component definitions with the newly read component definitions. New component definitions are added and deleted component definitions are removed.
<b>Step 3</b>	Read the Universal Access Control List configuration entries from the registry. Universal Broker replaces its UACL entries with the newly read entries.

#### Components

[Universal Control](#)

## Refreshing a Component from Windows

- Refreshing a Component via Universal Control from Windows
  - Command Line Options
  - Components

### Refreshing a Component via Universal Control from Windows

This example refreshes a component on a remote system.

```
uctl -refresh uems -cmdid "UEM-dallas" -host dallas -userid joe -pwd akkSdiq
```

### Command Line Options

The command line options used in this example are:

Option	Description
-refresh	Type of component to refresh on the remote system.
-cmd	Assigns a command identifier of " <b>UEM-dallas</b> " to the started component.
-host	Directs the command to a computer with a host name of <b>dallas</b> .
-userid	Remote user ID with which to execute the Universal Control Server process.
-pwd	Password for the user ID.

### Components

Universal Control

## Refreshing Universal Broker from UNIX

- Refreshing Universal Broker via Universal Control from UNIX
  - Command Line Options
  - Universal Broker Actions
  - Components

### Refreshing Universal Broker via Universal Control from UNIX

This example refreshes Universal Broker on a remote system.

```
uctl -refresh -host dallas -userid joe -pwd akkSdiq
```

#### Command Line Options

The command line options used in this example are:

Option	Description
-refresh	Instruction to refresh Universal Broker on the remote system.
-host	Directs the command to a computer with a host name of <b>dallas</b> .
-userid	Remote user ID with which to execute the Universal Control Server process.
-pwd	Password for the user ID.

#### Universal Broker Actions

This refresh request directs Universal Broker to take the following actions:

<b>Step 1</b>	Read its configuration file <b>ubroker.conf</b> . Universal Broker refreshes the following configuration options: <ul style="list-style-type: none"> <li>• MESSAGE_LANGUAGE</li> <li>• RUNNING_MAX</li> </ul>
<b>Step 2</b>	Read all component definitions found in the component definition directory. Universal Broker replaces all component definitions with the newly read component definitions. New component definitions are added and deleted component definitions are removed.
<b>Step 3</b>	Read the Universal Access Control List configuration file <b>uacl.conf</b> . Universal Broker replaces its UACL entries with the newly read entries.

#### Components

Universal Control

## Refreshing a Component from UNIX

- Refreshing a Component via Universal Control from UNIX
  - Command Line Options
  - Components

### Refreshing a Component via Universal Control from UNIX

This example refreshes a component on a remote system.

```
uctl -refresh uems -cmdid "UEM-dallas" -host dallas -userid joe -pwd akkSdiq
```

### Command Line Options

The command line options used in this example are:

Option	Description
<code>-refresh</code>	Type of component to refresh on the remote system.
<code>-cmdid</code>	Assigns a command identifier of <b>"UEM-dallas"</b> to the started component.
<code>-host</code>	Directs the command to a computer with a host name of <b>dallas</b> .
<code>-userid</code>	Remote user ID with which to execute the Universal Control Server process.
<code>-pwd</code>	Password for the user ID.

### Components

Universal Control

## Refreshing Universal Broker from IBM i

- Refreshing Universal Broker via Universal Control from IBM i
  - Command Line Options
  - Universal Broker Actions
  - Components

### Refreshing Universal Broker via Universal Control from IBM i

This example refreshes a component on a remote system.

```
STRUCT REFRESH(*yes) HOST(dallas) USERID(joe) PWD(akkSdiq)
```

#### Command Line Options

The command line options used in this example are:

Option	Description
REFRESH	Instruction to refresh Universal Broker on the remote system.
HOST	Directs the command to a computer with a host name of <b>dallas</b> .
USERID	Remote user ID with which to execute the Universal Control Server process.
PWD	Password for the user ID.

#### Universal Broker Actions

The REFRESH command directs Universal Broker to take the following actions:

<b>Step 1</b>	Read its configuration file <b>UNVCONF</b> and member <b>UBROKER</b> .
<b>Step 2</b>	Read all component definitions found in the component definition file, <b>UNVPRD510 / UNVCOMP</b> . Universal Broker replaces all component definitions with the newly read component definitions. New component definitions are added and deleted component definitions are removed.
<b>Step 3</b>	Read the Universal Access Control List configuration file <b>UNVCONF</b> and member <b>UACL</b> . Universal Broker replaces its UACL entries with the newly read entries.

#### Components

Universal Control

## Refreshing a Component from IBM i

- Refreshing a Component via Universal Control from IBM i
  - Command Line Options
  - Components

### Refreshing a Component via Universal Control from IBM i

This example refreshes a component on a remote system.

```
STRUCT REFRESH(*yes) RFSHCMPNM(uems) CMDID('UEM-dallas') HOST(dallas) USERID(joe) PWD(akkSdiq)
```

### Command Line Options

The command line options used in this example are:

Option	Description
REFRESH	Specification for whether or not to refresh.
RFSHCMPNM	Type of component to refresh on the remote system.
CMDID	Assigns a command identifier of <b>'UEM-dallas'</b> to the started component.
HOST	Directs the command to a computer with a host name of <b>dallas</b> .
USERID	Remote user ID with which to execute the Universal Control Server process.
PWD	Password for the user ID.

### Components

Universal Control

## Merging Configuration Options Examples

### Merging Configuration Options Examples

- [Files Used in UPI Merge Examples](#)
- [Merge Configuration Files Using Program Defaults](#)
- [Merge Configuration Files Introducing New Options](#)
- [Merge Configuration Files Using Installation-Dependent Values](#)

***These examples illustrate the merging of Workload Automation 5 (for Windows or UNIX) components' configuration options using the Universal Products Install Merge (UPI) component.***



## Files Used in UPI Merge Examples

- Files Used in Examples
  - Workload Automation 5 Configuration File Sample (infile.txt)
  - Workload Automation 5 Configuration File Sample (outfile.txt)

## Files Used in Examples

The examples in this section demonstrate the expected results when Universal Products Install Merge is executed using two configuration files with the contents identified in the following tables.



**Note**

Although these examples show Windows path names, the Universal Install Merge behavior demonstrated also applies to UNIX systems.

### Workload Automation 5 Configuration File Sample (infile.txt)

The following table identifies the contents of **infile.txt**, a sample file in the Workload Automation standard keyword / value configuration file format.

For the examples in this section, **infile.txt** could represent an existing or archived configuration file, or a work file used to introduce and distribute configuration values across one or more target systems.

Keyword	Value
installation_directory	"C:\Program Files\Universal\UCmdMgr"
message_level	info
#host	some.remote.host
port	7850
license_product	"UNIVERSAL COMMAND MANAGER"
license_customer	"STONEBRANCH, INC."
license_type	DEMO
license_expiration_date	2012.12.21
license_nt_servers	1
license_key	078B-E180-64E6-3016-EA20-0CF4-58F9-B301 *

\* This license key is for demonstration purposes only. It is not a valid license key.

**Workload Automation 5 Configuration File Sample (outfile.txt)**

The following table identifies the contents of **outfile.txt**, another sample file in the Workload Automation standard keyword / value configuration file format.

For the examples in this section, **outfile.txt** might represent a default configuration file that is delivered during product installation, or an existing production configuration file that needs to be updated with values from **infile.txt**.

Keyword	Value
port	7887
activity_monitoring	yes
event_generation	*,x100

## Merge Configuration Files Using Program Defaults

- Merge Configuration Files Using Program Defaults
  - Command Line Options
  - Merged File Contents
  - Components

### Merge Configuration Files Using Program Defaults

The following figure illustrates the command line used to merge configuration options from `infile.txt` into `outfile.txt`.

In this example, UPIMERGE executes using program defaults.

```
upimerge -dest outfile.txt -source infile.txt
```

### Command Line Options

The command line options used in this example are:

Option	Description
-dest	Name of a file used to store the result of the merge.
-source	Name of a file used as input to the merge. If this option is omitted, UPIMERGE assumes input is redirected via stdin.

### Merged File Contents

The following table identifies the contents of `outfile.txt` after UPIMERGE completes.

To obtain this result, UPIMERGE added options from `infile.txt` that did not exist in `outfile.txt` (that is, `installation_directory`, `message_level`, `license_key`, and so on). It also preserved the value for the `port` option by replacing the 7887 value with the currently defined 7850.

UPIMERGE also dropped the commented `host` option from `infile.txt`. UPIMERGE ignores any comments in the input file, because merging those lines into the output file would have no effect on the application's behavior.

Finally, UPIMERGE commented out the `activity_monitoring` and `event_generation` options introduced by `outfile.txt`. UPIMERGE cannot distinguish between options for new features and new values for existing options. To prevent the introduction of a new value into an application currently running with application-defined defaults, UPIMERGE's default response is to comment out any option in the output file with no match in the input file.

Keyword	Value
<code>installation_directory</code>	"C:\Program Files\Universal\UCmdMgr"
<code>message_level</code>	info
<code>port</code>	7850

license_product	"UNIVERSAL COMMAND MANAGER"
license_customer	"STONEBRANCH, INC."
license_type	DEMO
license_expiration_date	2012.12.21
license_nt_servers	1
license_key	078B-E180-64E6-3016-EA20-0CF4-58F9-B301
#activity_monitoring	yes
#event_generation	*,x100

## Components

[Universal Products Install Merge](#)

## Merge Configuration Files Introducing New Options

- [Merge Configuration Files Introducing New Options](#)
  - [Merged File Contents](#)
  - [Components](#)

### Merge Configuration Files Introducing New Options

The following figure illustrates the command line used to merge configuration options from `infile.txt` into `outfile.txt`.

In this example, UPIMERGE changes its default behavior, and introduces new values for the `activity_monitoring` and `event_generation` options by not commenting them out in the merged file.

```
upimerge -dest outfile.txt -source infile.txt -keep_nomatch yes
```

Option	Description
<code>-dest</code>	Name of a file used to store the result of the merge.
<code>-source</code>	Name of a file used as input to the merge. If this option is omitted, UPIMERGE assumes input is redirected via stdin.
<code>-keep_nomatch</code>	Controls merge behavior when an option in <code>-dest</code> has no match in <code>-source</code> .

### Merged File Contents

The following table identifies the contents of `outfile.txt` after UPIMERGE completes.

The result is almost identical to the example shown in [Merge Configuration Files Using Program Defaults](#). Executing UPIMERGE with `-keep_nomatch` set to `yes` enables the `activity_monitoring` and `event_generation` options in the output file.

Keyword	Value
<code>installation_directory</code>	"C:\Program Files\Universal\UCmdMgr"
<code>message_level</code>	info
<code>port</code>	7850
<code>license_product</code>	"UNIVERSAL COMMAND MANAGER"
<code>license_customer</code>	"STONEBRANCH, INC."
<code>license_type</code>	DEMO

license_expiration_date	2012.12.21
license_nt_servers	1
license_key	078B-E180-64E6-3016-EA20-0CF4-58F9-B301
activity_monitoring	yes
event_generation	*,x100

## Components

[Universal Products Install Merge](#)

## Merge Configuration Files Using Installation-Dependent Values

- Merge Configuration Files Using Installation-Dependent Values
  - Command Line Options
  - Merged File Contents
  - Components

### Merge Configuration Files Using Installation-Dependent Values

The following figure illustrates the command line used to merge configuration options from `infile.txt` into `outfile.txt`. In this example, UPIMERGE applies logic specific to a particular configuration file, and updates any references to locations that depend on the installed location of that Workload Automation application.

```
upimerge -dest outfile.txt -source infile.txt -cfgtype ucmd -installdir "D:\Program Files\Universal\UCmdMgr"
```

### Command Line Options

The command line options used in this example are:

Option	Description
<code>-dest</code>	Name of a file used to store the result of the merge.
<code>-source</code>	Name of a file used as input to the merge. If this option is omitted, UPIMERGE assumes input is redirected via stdin.
<code>-cfgtype</code>	Notifies UPIMERGE that <code>-source</code> is a configuration file that contains settings for the specified Workload Automation application.
<code>-installdir</code>	Primary location in which the Workload Automation application identified by <code>-cfgtype</code> resides.

### Merged File Contents

The following table identifies the contents of `outfile.txt` after UPIMERGE completes. The result is almost identical to the example shown in [Merge Configuration Files Using Program Defaults](#), except for the value of the `-installdir` option.

Even though `infile.txt` contained a value for `*-installdir*`, UPIMERGE interpreted that value as the application's current location. UPIMERGE then updated any values in `outfile.txt` (executing logic based on the specified `-cfgtype`) that depend on the installed location.

This example might be useful in a situation where it is necessary to recover configuration settings from an archived file, but the application no longer resides in the directory specified in the archive file. This is the logic that UPIMERGE uses during a Workload Automation installation to ensure that installation-dependent locations are always correct.

Keyword	Value
<code>installation_directory</code>	"D:\Program Files\Universal\UCmdMgr"
<code>message_level</code>	info

Port	7850
license_product	"UNIVERSAL COMMAND MANAGER"
license_customer	"STONEBRANCH, INC."
license_type	DEMO
license_expiration_date	2012.12.21
license_nt_servers	1
license_key	078B-E180-64E6-3016-EA20-0CF4-58F9-B301
#activity_monitoring	yes
#event_generation	*,x100

## Components

[Universal Products Install Merge](#)



# Infitran - Component Management

## Component Management - Overview

### Component Management

Component Management information for Infitran is comprised of:

- [Component Definition](#)
- [Component Definition Options](#)
- [Starting and Stopping Components](#)
- [Starting and Stopping Components - Examples](#)
- [Maintaining Universal Broker Definitions in UEC Database](#)

## Component Definition

### Overview

Each Infitran Server component (for Universal Data Mover, Universal Event Monitor, Universal Control, and Universal Application Container) has a Component Definition.

The Component Definition is a text file of options that defines component-specific information required by the Universal Broker.

Each Component Definition defines the following type of information:

- Component type (for Universal Event Monitor Servers only).
- Component name.
- Component command name.
- Component configuration file name.
- Component working directory path.
- Number of component instances that can run simultaneously.
- Specification for whether or not the component starts automatically when the Universal Broker starts.

The reference guide for each component contains detailed information about its Component Definition.

### Universal Event Monitor Component Definition

The Component Definition for a Universal Event Monitor Server defines whether it is a demand-driven or an event-driven server. Among other factors, this determines how the server is started (see [Starting and Stopping Components](#)).

For a complete explanation of the difference between demand-driven and event-driven Universal Event Monitor Servers, see [UEM Servers - Demand-Driven vs. Event-Driven](#).

## Component Definition Options

### Infitran Component Definition Options

The following component definition options are available for Infitran components:

[Universal Broker Component Definition Options](#)

[Universal Automation Center Agent Component Definition Options](#)

[Universal Command Component Definition Options](#)

[UAC Server Component Definition Options](#)

[Universal Data Mover Component Definition Options](#)

[Universal Event Monitor Component Definition Options](#)

[Universal Control Component Definition Options](#)

## Starting and Stopping Components

- Starting Components
  - Starting Manually
  - Starting via Manager
  - Starting Automatically
  - Starting via Universal Control
- Stopping Components

### Starting Components

There are four ways in which Infitran components are started.

#### Starting Manually

The following components are started manually and run in the background until they are stopped manually:

- Universal Broker
- Universal Enterprise Controller

(See [Starting and Stopping Components - Examples.](#))

#### Starting via Manager

The following components are started on demand (that is, via their Managers) and run until the specified task has completed, then stop automatically.

- Universal Data Mover Server
- Universal Control Server
- Universal Event Monitor Server (demand-driven)

#### Starting Automatically

The following components are auto-start components; that is, they start automatically when the Universal Broker starts and run until they are stopped manually:

- Universal Application Container Server
- Universal Event Monitor Server (event-driven)
- Universal Automation Center Agent



**Note**

A Universal Event Monitor Server Component Definition also can specify that an event-driven server is not started automatically (see [Starting via Universal Control](#), below).

#### Starting via Universal Control

Universal Control can start Server components, via the `START_CMD` that do not require interaction with a Manager. Currently, only two Infitran components can be started via Universal Control:

- Universal Event Monitor Server (event-driven)
- Universal Automation Center Agent

(See [Starting and Stopping Components - Examples.](#))

### Stopping Components

Any Infitran Server component can be stopped via the Universal Control `STOP_CMD` option.

Authorized users also are able to use the I-Activity Monitor, a Universal Enterprise Controller (UEC) client application, to stop running any Infitran Server component (if it is a component of an Agent being polled by UEC).

## Starting and Stopping Components - Examples

- Starting and Stopping Universal Broker Examples
- Starting and Stopping Universal Enterprise Controller Examples
- Starting and Stopping Components via Universal Control Examples

### Starting and Stopping Universal Broker Examples

- Starting and Stopping Universal Broker for z/OS
- Starting Universal Broker for Windows
- Starting Universal Broker for UNIX
- Starting, Ending, and Working with Universal Broker for IBM i

### Starting and Stopping Universal Enterprise Controller Examples

- Starting and Stopping Universal Enterprise Controller for z/OS
- Starting and Stopping Universal Enterprise Controller for Windows

### Starting and Stopping Components via Universal Control Examples



**Note**

Currently, only Universal Event Monitor Servers and Universal Automation Center Agent can be started by Universal Control.

The examples assume that Universal Control Server is installed on a remote system named **dallas**. The user ID and password used in the examples must be changed to a valid user ID and password for the remote system.

Links to detailed technical information on appropriate Indesca components are provided for each example.

- Starting a z/OS Component via Universal Control
- Stopping a z/OS Component via Universal Control
- Starting a Windows Component via Universal Control
- Stopping a Windows Component via Universal Control
- Starting a UNIX Component via Universal Control
- Stopping a UNIX Component via Universal Control
- Starting an IBM i Component via Universal Control
- Stopping an IBM i Component via Universal Control



**Note**

These examples reference the IBM i commands by their untagged names. If you are using commands with tagged names to run **Universal Broker** and **Universal Control**, substitute the tagged names for these untagged names. (For information on tagged names, see UCHGRS (Change Release Tag) Program.)

## Starting and Stopping Universal Broker for zOS

### Overview

Universal Broker for z/OS executes as a started task.

The UBROKER program utilizes the z/OS UNIX System Services environment.

### Start Universal Broker

To start Universal Broker, execute the **START** console command:

```
START UBROKER[,UPARM='options']
```

### Stop Universal Broker

To stop Universal Broker, execute the **STOP** console command:

```
STOP UBROKER
```

## Starting Universal Broker for Windows

- [Overview](#)
- [Console Application](#)
  - [Console Security](#)
- [Windows Service](#)
  - [Service Security](#)
  - [Required File System Permissions](#)
  - [Executing the Broker Service With a Domain Account](#)

### Overview

Universal Broker can be executed in two different environments:

- Console application
- Windows service

### Console Application

The **ubroker** command starts Universal Broker as a console application.

Enter **ubroker** either from the:

- Command Prompt window
- **Run** dialog (Select **Run...** from the Windows **Start** menu.)

### Console Security

Universal Broker inherits its user account from the user that starts it. The Broker itself does not require any additional permissions or rights other than the default ones granted to the Windows group user.

However, components started by the Broker also run with the same user account as the Broker. Some components may require permissions or rights other than those granted to the user account that started the Broker.

For additional information regarding the security requirements of Universal Broker and all Indesca components, see [Workload Automation 5 - Security](#).

### Windows Service

Universal Broker is installed as a Windows service that starts automatically when the system is started. Windows provides a utility called **Services** that is used to interact with and manage all installed services. **Services** is an item in the Administrative Tools program group, which is accessible from the Control Panel.

### Service Security

The Universal Broker service can be configured to execute with the Local System account or with a specially configured Administrative account. The Local System account automatically provides the permissions necessary to execute the Broker.

An administrative account must have the following privileges to execute the Broker:

- Act as part of the operating system
- Adjust memory quotas for a process
- Bypass traverse checking
- Debug programs
- Log on as a service
- Impersonate a client after authentication
- Increase scheduling priority
- Replace a process level token
- Take ownership of files and other objects

To restrict interactive access by the account to the system, we also recommend adding the following policies:

- Deny log on as batch job
- Deny log on locally
- Deny log on through Terminal Services



Any existing Administrative account may be configured as described above to execute the Broker. The Workload Automation install also provides the ability to create and configure an Administrative account with the privileges above.

Configuring the Broker to run with an Administrative account not only allows the service to execute with just the privileges it needs, it also enables the Broker service to access network resources it would not have visibility to while executing as Local System.

### Required File System Permissions

It may be necessary to update the Broker account's access to the Workload Automation installed directories and files. If the product is installed to its default location under the Program Files directory, the local Administrative account used to execute the Broker (such as the default **UBrokerService** account) will likely get the file system access it needs via permissions inherited from parent directories.

However, if the application is installed to a location outside of the Program Files path – or a domain account is used to execute the Broker Service – the required file system permissions may need to be added after the install.

The recommended approach is to grant the Broker service account Full Control of the following directories, making sure that the permissions are propagated to all sub-directories and files:

- **.\Universal** install directory.
- **%ALLUSERSPROFILE%\Application Data\Universal** directory, which is the parent directory of the **.\conf** and **.\comp** directories in which the configuration files and component definition files reside, respectively.

Full control is recommended because of the varied requirements and configurations possible with the Workload Automation components. However, should you desire a more precise configuration, the Broker user only requires Read/Execute permissions for the following directories, along with their sub-directories and files:

- **.\Universal\Inls**
- **.\Universal\UCmdMgr**
- **.\Universal\UCtlMgr**
- **.\Universal\UDMMgr**
- **.\Universal\UEId**
- **.\Universal\UEMMgr**
- **.\Universal\UPIMerge**
- **.\Universal\UQuery**
- **.\Universal\USpool**



#### Note

The Workload Automation installation itself does not set the required file permissions for the Broker user. It only relies on permissions inherited from parent directories.

### Executing the Broker Service With a Domain Account

The Universal Broker service may be configured to run with a Windows domain account. To do so, simply make sure the account already exists before starting the installation (the Workload Automation install will not configure a domain account) and verify that the account has the [privileges](#) and [file system access](#) listed above.

## Starting Universal Broker for UNIX

- Starting Universal Broker for UNIX
- Console Application
  - Console Security
- Daemon
  - Daemon Security

### Starting Universal Broker for UNIX

Universal Broker can be executed in two different environments:

- Console Application
- Daemon

Differences between the environments are described in the following sections.

Only one instance of the Universal Broker can execute at any one time. A PID file is used to help ensure that there is only one active instance; it is a locking mechanism that prevents the execution of a second Broker. The PID file, **ubroker.pid**, is created in directory **/var/opt/universal** by default. If the PID file is in the PID directory, it is assumed that a Broker instance is executing.

### Console Application

The **ubroker** command starts Universal Broker as a console application.

#### Console Security

Universal Broker runs with the same user ID as the user who starts it. The Broker does not require superuser rights. It only requires access to its installation directory and files, which often are created by the superuser account when the product is installed.

However, components started by the Broker also run with the same user ID as the Broker. Some of these components may require superuser rights.

See [Workload Automation 5 - Security](#) for details on their security requirements for specific Workload Automation 5 components.

### Daemon

Universal Broker can run as a UNIX daemon process. This is the preferred method of running the Broker. A daemon start-up script is provided to manage the starting and stopping of the Broker daemon. The startup script utilizes the PID file to ensure that only one instance of the Broker is executing at any one time. For this reason, the start-up script should be used to start and stop the Broker.



#### Note

Although they have the same name, the Broker daemon start-up script should not be confused with the actual Broker daemon program file.

- Startup script is installed in the primary Broker directory (that is, **./universal/ubroker**).
- Program file is installed in the Broker's **bin** directory (that is, **./universal/ubroker/bin**).

```
ubrokerd { start | stop | status | restart }
```

The following table describes the command line arguments to the Universal Broker daemon start-up script.

Command	Description
start	Starts the Universal Broker daemon. Only one instance of Universal Broker can run at any given time, so if the Broker

	already is running, the command fails and the script returns.
stop	Stops the Universal Broker daemon. If the Broker daemon is not running, the script simply returns.
status	Returns the status of the Universal Broker daemon, either <i>running</i> or <i>stopped</i> . If the daemon is running, the script displays its process ID.
restart	Performs a <b>stop</b> request followed by a <b>start</b> request.

## Daemon Security

When a daemon is started at system initialization, it is started as user **root**. The root user ID provides sufficient authority for the Broker and any component it may start.

If the daemon is started with a non-root user ID, the environment is the same as if it was started as a console application. (See [Console Security](#), above, for more details.)

## Starting, Ending, and Working with Universal Broker for IBM i

- Starting, Ending, and Working With Universal Broker for IBM i
- Commands
  - Start Subsystem Command (STRSBS)
  - End Subsystem Command (ENDSBS)
  - Work With Subsystem Command (WRKSBS)

### Starting, Ending, and Working With Universal Broker for IBM i

Universal Broker executes within its own IBM i subsystem, named **UNVUBR510**. The **UNVUBR510** subsystem provides a self-contained environment in which Universal Broker can be managed. The **UNVUBR510** subsystem description (object type **\*SBSD**) is named **UNVUBR510**.

The **UNVUBR510** subsystem contains several entries that define the subsystem environment. The two most visible are:

- Autostart entry
- Pre-start job entries

The subsystem autostart entry defines what jobs are started automatically when the subsystem is started. The **UNVUBR510** subsystem defines one autostart entry, **UNVUBR510**. The **UBROKER** job executes with the job description **UBROKER** (object type **\*JOB**) and user profile **UNVUBR510** (object type **\*USRPRF**). Only one instance of the **UBROKER** job, which runs continuously, can be active at any one time within the context of any one Stonebranch-defined subsystem.

The subsystem pre-start job entries define jobs that are in an initialized state. They are not executing but are ready to accept a request and execute at any time. Pre-starting jobs before they are required improves the overall throughput of the subsystem jobs.

Universal Broker jobs running under **UNVUBR510** use the **UBROKER** job queue and class located in the product installation library. See the [IBM i Installation - Customization](#) for additional information.

The Universal Command (UCMD) Server jobs log all significant events to the **UBROKER** job log. However, by default, IBM i does not keep job logs unless the job terminates due to an error. As a result, important information relevant to server errors may be discarded when the **UBROKER** job is shut down normally.

To preserve the server-related information, the **UBROKER** job description specifies Message Logging as `4 0 *MSG`. The **UBROKER** job's job log will be sent automatically to the output queue and printer device designated in the **UBROKER** job description, which is located in the Workload Automation installation library, **UNVPRD510** (by default).

In some very large organizations with heavy **UBROKER** usage, the job log may fill. By default, IBM i jobs are stopped when the job log fills. To ensure continuous **UBROKER** operation, Workload Automation sets the job log to wrap. (See [IBM i Installation](#) for additional information.)

### Commands

The following O/S commands help manage the **UNVUBR510** subsystem.

#### Start Subsystem Command (STRSBS)

Starts the Universal Broker subsystem, **UNVUBR510**.

```
STRSBS UNVPRD510/UNVUBR510
```

#### End Subsystem Command (ENDSBS)

Ends the Universal Broker subsystem, **UNVUBR510**.

```
ENDSBS UNVUBR510
```

#### Work With Subsystem Command (WRKSBS)

Allows users to work with all active subsystems. Choose the **UNVUBR510** subsystem from the list of subsystems displayed.

WRKSBS

## Starting and Stopping Universal Enterprise Controller for zOS

- Overview
- Starting UEC
- Stopping UEC
- System MODIFY Command
  - DUMP Command
  - BROKERSTAT Command

### Overview

Universal Enterprise Controller (UEC) for z/OS executes as a started task.

### Starting UEC

The UEC started task, **UECTLR**, is started with the z/OS START command:

```
*S UECTLR*
```

### Stopping UEC

The UEC started task, **UECTLR**, is stopped with the z/OS MODIFY STOP command:

```
*P UECTLR*
```

After the STOP command is issued, UEC may take several seconds to shut down.



#### Note

The **UECTLR** started task should run at a high dispatch priority in order to avoid not being dispatched in a timely enough manner to process the agent polling protocol. If **UECTLR** is not dispatched appropriately, the Broker may be reported as timed out when the Broker itself still is operational.

### System MODIFY Command

The UEC started task accepts commands via the system MODIFY command. The MODIFY command's **APPL=** parameter is required, since UEC runs as a USS address space.

#### DUMP Command

The DUMP command directs UEC to produce a Language Environment dump. The dump is written to the **CEEDUMP** ddname. While the dump is being produced, UEC is paused by LE until the dump completes, after which UEC continues processing.

In the following example, the procedure name **UECTLR** is assumed:

```
*F UECTLR,APPL=DUMP*
```

The DUMP command is used for diagnostic purposes. It should be executed only at the request of Stonebranch, Inc.

#### BROKERSTAT Command

The BROKERSTAT command provides on-demand Broker status alerting. It causes UEC to issue an alert message for all defined Brokers indicating their current internal state.

- Alert UNV1056T (Unable to connect) is issued for Brokers that are down.
- Alert UNV1059T (Broker responding) is issued for Brokers that are up.

The alert message is equivalent to what UEC issued at the time the alert was originally generated.

In the example below, the procedure name **UECTLR** is assumed:

```
*F UECTLR,APPL=BROKERSTAT*
```

Alerts issued on-demand (by BROKERSTAT) are not sent to the I-Activity Monitor client. (When issued under normal processing by UEC, the alerts are sent to I-Activity Monitor.)

## Starting and Stopping Universal Enterprise Controller for Windows

### Starting / Stopping Universal Enterprise Controller for Windows

Universal Enterprise Controller (UEC) for Windows executes as a service.

By default, UEC for Windows is set to start automatically whenever Windows is booted.

Changes to UEC configuration requires it to be stopped and restarted by the Windows Service Control Manager.

To access the Service Control Manager:

<b>Step 1</b>	Click the <b>Control Panel</b> on the Windows Start menu.
<b>Step 2</b>	Double-click the <b>Administrative Tools</b> icon on the Control Panel window.
<b>Step 3</b>	Double-click the <b>Services</b> icon on the Administrative Tools window.
<b>Step 4</b>	On the Services window, select Universal Enterprise Controller in the list of services.
<b>Step 5</b>	In the Action menu, click: <ol style="list-style-type: none"><li>1. Stop, to stop UEC for Windows.</li><li>2. Start, to start UEC for Windows.</li></ol>



## Starting a z/OS Component via Universal Control

- Starting a z/OS Component via Universal Control
  - SYSIN Options
  - Components

### Starting a z/OS Component via Universal Control

This example – located in the Universal Control **SUNVSAMP** library – starts a component on a remote system.

It assumes that Universal Control Server is installed on a remote system named **dallas**. The user ID and password used in the example must be changed to a valid user ID and password for the remote system.

```
//jobname JOB CLASS=A,MSGCLASS=X
//STEP1 EXEC UCTLPRC
//SYSIN DD *
-start uems -cmdid "UEM-dallas" -host dallas -userid joe -pwd akkSdiq
/*
```

### SYSIN Options

The SYSIN options used in this example are:

Option	Description
<code>-start</code>	Name of the component to start on the remote system.
<code>-cmdid</code>	Assigns a command identifier of " <b>UEM-dallas</b> " to the started component.
<code>-host</code>	Directs the command to a computer with a host name of <b>dallas</b> .
<code>-userid</code>	Remote user ID with which to execute the Universal Control Server process. The started component, in fact, will execute with the Universal Broker's security context.
<code>-pwd</code>	Password for the user ID.

### Components

Universal Control

## Stopping a zOS Component via Universal Control

- [Stopping a zOS Component via Universal Control](#)
- [SYSIN Options](#)
- [Components](#)

### Stopping a zOS Component via Universal Control

This example – located in the Universal Control **SUNVSAMP** library – stops a component on a remote system.

It assumes that Universal Control Server is installed on a remote system named **dallas**. The user ID and password used in the example must be changed to a valid user ID and password for the remote system.

```
//jobname JOB CLASS=A,MSGCLASS=X
//STEP1 EXEC UCTLPRC
//SYSIN DD *
-stop 999234133 -host dallas -userid joe -pwd akkSdiq
/*
```

The sample JCL is located in member **UCTSAM1**.

The JCL procedure **UCTLPRC** is used to execute the stop request.

The stop request is sent to a remote system named **dallas** for execution.

### SYSIN Options

The SYSIN options used in this example are:

Option	Description
<a href="#">-stop</a>	Component to stop.
<a href="#">-host</a>	Directs the command to a computer with a host name of <b>dallas</b> .
<a href="#">-userid</a>	Remote user ID with which to execute the stop request.
<a href="#">-pwd</a>	Password for the user ID.

### Components

[Universal Control](#)

## Starting a Windows Component via Universal Control

- [Starting a Windows Component via Universal Control](#)
  - [Command Line Options](#)
  - [Components](#)

### Starting a Windows Component via Universal Control

This example starts a component on a remote system.

```
uctl -start uems -cmdid "UEM-dallas" -host dallas -userid joe -pwd akkSdiq
```

### Command Line Options

The command line options used in this example are:

Option	Description
<code>-start</code>	Name of the component to start on the remote system.
<code>-cmdid</code>	Assigns a command identifier of <b>"UEM-dallas"</b> to the started component.
<code>-host</code>	Directs the command to a computer with a host name of <b>dallas</b> .
<code>-userid</code>	Remote user ID with which to execute the Universal Control Server process. The started component, in fact, will execute with the Universal Broker's security context.
<code>-pwd</code>	Password for the user ID.

### Components

[Universal Control](#)

## Stopping a Windows Component via Universal Control

- [Stopping a Windows Component via Universal Control](#)
  - [Command Line Options](#)
  - [Components](#)

## Stopping a Windows Component via Universal Control

This example stops a component on a remote system.

```
uctl -stop 10739132 -host dallas -userid joe -pwd akkSdiq
```

### Command Line Options

The command line options used in this example are:

Option	Description
<code>-stop</code>	Component to stop.
<code>-host</code>	Directs the command to a computer with a host name of <b>dallas</b> .
<code>-userid</code>	Remote user ID with which to execute the stop request.
<code>-pwd</code>	Password for the user ID.

### Components

[Universal Control](#)

## Starting a UNIX Component via Universal Control

- [Starting a UNIX Component via Universal Control](#)
  - [Command Line Options](#)
  - [Components](#)

### Starting a UNIX Component via Universal Control

This example starts a component on a remote system.

```
uctl -start uems -cmdid "UEM-dallas" -host dallas -userid joe -pwd akkSdiq
```

### Command Line Options

The command line options used in this example are:

Option	Description
<code>-start</code>	Name of the component to start on the remote system.
<code>-cmdid</code>	Assigns a command identifier of <b>"UEM-dallas"</b> to the started component.
<code>-host</code>	Directs the command to a computer with a host name of <b>dallas</b> .
<code>-userid</code>	Remote user ID with which to execute the Universal Control Server process. The started component, in fact, will execute with the Universal Broker's security context.
<code>-pwd</code>	Password for the user ID.

### Components

[Universal Control](#)

## Stopping a UNIX Component via Universal Control

- [Stopping a UNIX Component via Universal Control](#)
  - [Command Line Options](#)
  - [Components](#)

## Stopping a UNIX Component via Universal Control

This example stops a component on a remote system.

```
uctl -stop 10739132 -host dallas -userid joe -pwd akkSdiq
```

### Command Line Options

The command line options used in this example are:

Option	Description
<code>-stop</code>	Component to stop.
<code>-host</code>	Directs the command to a computer with a host name of <b>dallas</b> .
<code>-userid</code>	Remote user ID with which to execute the stop request.
<code>-pwd</code>	Password for the user ID.

### Components

Universal Control

## Starting an IBM i Component via Universal Control

- [Starting an IBM i Component via Universal Control](#)
  - [Command Line Options](#)
  - [Components](#)

### Starting an IBM i Component via Universal Control

This example starts a component on a remote system.

```
STRUCT START(uems) CMDID('UEM-dallas') HOST(dallas) USERID(joe) PWD(akkSdiq)
```



#### Note

This example references the IBM i command by its untagged name. If you are using commands with tagged names to run Universal Control, substitute the tagged names for the untagged names.

### Command Line Options

The command line options used in this example are:

Option	Description
START	Component to start on the remote system.
CMDID	Assigns a command identifier of 'UEM-dallas' to the started component.
HOST	Directs the command to a computer with a host name of <b>dallas</b> .
USERID	Remote user ID with which to execute the Universal Control Server process. The started component, in fact, will execute with the Universal Broker's security context.
PWD	Password for the user ID.

### Components

Universal Control

## Stopping an IBM i Component via Universal Control

- [Stopping an IBM i Component via Universal Control](#)
  - [Command Line Options](#)
  - [Components](#)

### Stopping an IBM i Component via Universal Control

This example stops a component on a remote system.

```
STRUCT STOP(10739132) HOST(dallas) USERID(joe) PWD(akkSdiq)
```



#### Note

This example references the IBM i command by its untagged name. If you are using commands with tagged names to run Universal Control, substitute the tagged names for the untagged names.

### Command Line Options

The command line options used in this example are:

Option	Description
STOP	Component on the remote system to stop.
HOST	Directs the command to a computer with a host name of <b>dallas</b> .
USERID	Remote user ID with which to execute the stop request. This must match the user ID originally used to start the component.
PWD	Password for the user ID.

### Components

[Universal Control](#)



## Maintaining Universal Broker Definitions in UEC Database

### Maintaining Universal Broker Definitions in UEC Database

- [Maintaining Broker Definitions in UEC Database - zOS and Windows](#)
- [Maintaining Broker Definitions in UEC Database - zOS](#)
- [Maintaining Broker Definitions in UEC Database - Windows](#)



**Note**

All of the tasks illustrated on these pages are implemented with use of the [UECLoad Utility](#) component.

## Maintaining Broker Definitions in UEC Database - zOS and Windows

- List All Defined Universal Brokers
  - Command Line Options
- Export a Specific, Defined Universal Broker
  - Command Line Options
- Export Events
  - Command Line Options
- Delete a Specific, Defined Universal Broker
  - Command Line Options
- Add Specific, Defined Universal Broker via deffile
  - Command Line Options
  - Definition File
- Add Existing Universal Brokers to a Broker Group
  - Command Line Options
- Delete Existing Universal Brokers from a Broker Group
  - Command Line Options

### List All Defined Universal Brokers

The following illustrates the output of a user-friendly format of the Universal Brokers defined in the UEC database.

```
uecload -port 8778 -userid joe -pwd akkSdiq -list -broker_name "**"
```

### Command Line Options

The command line options used in this example are:

Option	Description
-port	TCP/IP port number of the UEC.
-userid	UEC user ID/account with which Brokers will be modified.
-pwd	Password associated with -userid.
-list	Output the described broker definition in a user-friendly format.
-broker_name	"**" specifies all Universal Brokers.

### Export a Specific, Defined Universal Broker

The following illustrates the output of a Universal Broker defined in the UEC database in a format suitable for use within a broker definition file.

```
uecload -port 8778 -userid joe -pwd akkSdiq -level audit -export -broker_name mybroker1
```

### Command Line Options

The command line options used in this example are:

Option	Description
-port	TCP/IP port number of the UEC.
-userid	UEC user ID/account with which Brokers will be modified.
-pwd	Password associated with -userid.
-level	Level of messages written.
-export	Output the described broker definition in a format to be used by a broker definition file.
-broker_name	Unique name of the defined Universal Broker.

## Export Events

The following illustrates the export of an events file into CSV format.

```
uecload -port 8778 -userid joe -pwd akkSdiq -level audit -export EVENTS -stime "*-5" -etime "*"
-format CSV -deffile events.csv
```



### Note

The double quotation marks ( " ) are required only with UNIX.

## Command Line Options

The command line options used in this example are:

Option	Description
-port	TCP/IP port number of the UEC.
-userid	UEC user ID/account with which Brokers will be modified.
-pwd	Password associated with -userid.
-level	Level of messages written.
-export	Output the described broker definition in a format to be used by a broker definition file.
-stime	Start time of exported data.

<code>-etime</code>	End time of exported data.
<code>-format</code>	Format of the output from the <code>-export EVENTS</code> action.
<code>-deffile</code>	File containing multiple broker definitions to be added or deleted in the UEC database.

## Delete a Specific, Defined Universal Broker

The following figure illustrates the deletion of a Universal Broker defined in the UEC database. Specifically, Universal Broker **mybroker1** is deleted from use of UEC.

```
uecload -port 8778 -userid joe -pwd akkSdiq -level audit -delete -broker_name mybroker1
```

## Command Line Options

The command line options used in this example are:

Option	Description
<code>-port</code>	TCP/IP port number of the UEC.
<code>-userid</code>	UEC user ID/account with which Brokers will be modified.
<code>-pwd</code>	Password associated with <code>-userid</code> .
<code>-level</code>	Level of messages written.
<code>-delete</code>	Delete Agent definitions from UEC.
<code>-broker_name</code>	Unique name of the defined Universal Broker.

## Add Specific, Defined Universal Broker via deffile

The following figure illustrates the addition of a group of Universal Broker definitions specified within a definition file in the UEC database. The name **sample\_deffile** represents the name of the created file.

```
uecload -port 8778 -userid joe -pwd akkSdiq -level audit -add -deffile sample_deffile
```

## Command Line Options

The command line options used in this example are:

Option	Description

-port	TCP/IP port number of the UEC.
-userid	UEC user ID/account with which Brokers will be modified.
-pwd	Password associated with -userid.
-level	Level of messages written.
-add	Add Agent definitions to UEC.
-deffile	File containing multiple broker definitions to be added or deleted in the UEC database.

### Definition File

The following figure is the definition file to be used for this example.

```
<BROKERDEF>
broker_name mybroker1
broker_host localhost
broker_port 7887
broker_desc "This is a description of broker1."
groups "Group 1, Group 2,Group 3"
<BROKERDEF>
<BROKERDEF>
broker_name mybroker2
broker_host 127.0.0.1
broker_port 7887
broker_desc "This is a description of broker2."
groups "Group 1, Group 2, Group 3"
<BROKERDEF>
<BROKERDEF>
broker_name mybroker3
broker_host 10.20.30.40
broker_port 7887
broker_desc "This is a description of broker3."
groups "Group 1, Group 2, Group 3"
<BROKERDEF>
```

### Add Existing Universal Brokers to a Broker Group

The following illustrates the addition of existing Universal Brokers to a Broker group.

```
uecload -port 8778 -userid joe -pwd akkSdiq -add -deffile brokers -groups "Test 1, Test 2, Test 3"
```

### Command Line Options

The command line options used in this example are:

Option	Description
-port	TCP/IP port number of the UEC.
-userid	UEC user ID/account with which Brokers will be modified.

<code>-pwd</code>	Password associated with <code>-userid</code> .
<code>-add</code>	Add Agent definitions to specified group(s).
<code>-deffile</code>	File containing multiple broker definitions to be added or deleted in the UEC database.
<code>-groups</code>	Group(s) in which the defined Universal Broker is a member. The Universal Broker will be added to the Group(s).

## Delete Existing Universal Brokers from a Broker Group

The following illustrates the deletion of existing Universal Brokers from a Broker group.

```
uecload -port 8778 -userid joe -pwd akkSdiq -delete -deffile brokers -groups "Test 2, Test 3"
```

### Command Line Options

The command line options used in this example are:

Option	Description
<code>-port</code>	TCP/IP port number of the UEC.
<code>-userid</code>	UEC user ID/account with which Brokers will be modified.
<code>-pwd</code>	Password associated with <code>-userid</code> .
<code>-delete</code>	Delete Agent definitions from specified group(s).
<code>-deffile</code>	File containing multiple broker definitions to be added or deleted in the UEC database.
<code>-groups</code>	Group(s) in which the defined Universal Broker is a member. The Universal Broker will be added to the Group(s).

## Maintaining Broker Definitions in UEC Database - z/OS

- Export Events into ARC Format for z/OS
  - SYSIN Options
- Retrieve Archived File and Export into XML for z/OS
  - SYSIN Options

### Export Events into ARC Format for z/OS

The following figure illustrates the export of events into an ARC format file on z/OS.

```
//STEP1 EXEC PGM=UECLOAD,PARM='ENVAR(TZ=EST5EDT) / '
//STEPLIB DD DISP=SHR,DSN=#HLQ.UNV.SUNVLOAD
//*
//UNVCONF DD DISP=SHR,DSN=#HLQ.UNV.UNVCONF(UECCFG00)
//*
//UNVTRACE DD SYSOUT=*
//ARCFILE DD DSN=APP.UEC.ARCH,
//          DISP=(,CATLG),UNIT=3390,VOL=SER=STG001,
//          SPACE=(CYL,(5,5)),
//          DCB=(RECFM=FB,LRECL=200,BLKSIZE=8000)
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
//SYSIN DD *
-export EVENTS -port 8778 -userid joe -pwd akkSdiq -level audit
-stime 2008/04/29,10:00:00 -etime 2008/04/30,10:00:00
-format ARC -deffile ARCFILE
```

### SYSIN Options

The SYSIN options used in this example are:

Option	Description
-export	Output the described broker definition in a format to be used by a broker definition file.
-port	TCP/IP port number of the UEC.
-userid	UEC user ID/account with which Brokers will be modified.
-pwd	Password associated with -userid.
-level	Level of messages written.
-stime	Start time of exported data.
-etime	End time of exported data.
-format	Format of the output from the -export EVENTS action.

`-deffile`

File containing multiple broker definitions to be added or deleted in the UEC database.

## Retrieve Archived File and Export into XML for z/OS

The following figure illustrates the retrieval of an archived file and its export into XML on z/OS.

```
//STEP1      EXEC PGM=UECLOAD,PARM='ENVAR(TZ=EST5EDT) / '
//STEPLIB   DD  DISP=SHR,DSN=#HLQ.UNV.SUNVLOAD
// *
//UNVCONF   DD  DISP=SHR,DSN=#HLQ.UNV.UNVCONF(UECCFG00)
//OUTPUT    DD  SYSOUT=*
//UNVTRACE  DD  SYSOUT=*
//ARCFILE   DD  DSN=APP.UEC.ARCH,DISP=SHR
//DEFFILE   DD  DSN=APP.UEC.DEFFILE,DISP=SHR
//SYSOUT    DD  SYSOUT=*
//CEEDUMP   DD  SYSOUT=*
//SYSIN     DD  *
-export EVENTS -arcfile ARCFILE -level audit
-format XML -deffile DEFFILE
```

## SYSIN Options

The SYSIN options used in this example are:

Option	Description
<code>-export</code>	Output the described broker definition in a format to be used by a broker definition file.
<code>-arcfile</code>	Archived file to retrieve for export.
<code>-level</code>	Level of messages written.
<code>-format</code>	Format of the output from the <code>-export</code> EVENTS action.
<code>-deffile</code>	File containing multiple broker definitions to be added or deleted in the UEC database.



## Maintaining Broker Definitions in UEC Database - Windows

### Export Events into ARC Format for Windows

The following illustrates the export of events into an ARC format file on Windows.

```
uecload -export EVENTS -userid admin -pwd admin -format ARC -stime 2011/06/24 -etime 2011/07/24
-deffile c:\test.xml -arcfile c:\test.arc
```

### Command Line Options

The command line options used in this example are:

Option	Description
-export	Output the described broker definition in a format to be used by a broker definition file.
-userid	UEC user ID/account with which Brokers will be modified.
-pwd	Password associated with -userid.
-format	Format of the output from the -export EVENTS action.
-stime	Start time of exported data.
-etime	End time of exported data.
-deffile	File containing multiple broker definitions to be added or deleted in the UEC database.
-arcfile	Archived file to retrieve for export.

### Retrieve Archived File and Export into CSV for Windows

The following illustrates the retrieval of an archived file and its export into CSV on Windows.

```
uecload -arcfile c:\test.arc -export EVENTS -stime 2011/10/07 -etime 2012/01/01 -level audit -format
CSV -deffile c:\test.csv
```



#### Note

**-port**, **-userid**, and **-pwd** are not used, since no connection is made to UEC for this operation.

### Command Line Options

The command line options used in this example are:

Option	Description
<code>-arcfile</code>	Archived file to retrieve for export.
<code>-export</code>	Output the described broker definition in a format to be used by a broker definition file.
<code>-stime</code>	Start time of exported data.
<code>-etime</code>	End time of exported data.
<code>-level</code>	Level of messages written.
<code>-format</code>	Format of the output from the <code>-export</code> EVENTS action.
<code>-deffile</code>	File containing multiple broker definitions to be added or deleted in the UEC database.

## **Infitran - Messaging and Auditing**

## **Messaging and Auditing - Overview**

### **Messaging and Auditing**

All Workload Automation 5 components have the same message facilities. Messages — in this context — are text messages written to a console, file, or system log that:

1. Document the actions taken by a program.
2. Inform users of error conditions encountered by a program.

These pages describe the message and audit facilities that are common to all Workload Automation 5 components. (See the individual Workload Automation 5 documentation for detailed technical information.)

## Messaging

- [Workload Automation 5 Messaging](#)
- [Message Types](#)
- [Message ID](#)
- [Message Levels](#)
- [Message Destinations](#)
  - [z/OS Message Destinations](#)
  - [Windows Message Destinations](#)
  - [UNIX Message Destinations](#)
  - [IBM i Message Destinations](#)
  - [HP NonStop Message Destinations](#)

### Workload Automation 5 Messaging

This page describes the Workload Automation 5 messaging facility:

#### Message Types

There are six types (or severity levels) of Workload Automation 5 messages. (The severity level is based on the type of information provided by those messages.)

<b>Audit</b>	Document the configuration options used by the program's execution and resource allocation details. They provide complete description of the program execution for auditing and problem resolution.
<b>Informational</b>	Document the actions being taken by a program. They help determine the current stage of processing for a program. They also document statistics about data processed.
<b>Warning</b>	Document unexpected behavior that may cause or indicate a problem.
<b>Error</b>	Document program errors. They provide diagnostic data to help identify the cause of the problem.
<b>Diagnostic</b>	Document diagnostic information for problem resolution.
<b>Alert</b>	Document a notification that a communications issue, which does not disrupt the program or require action, has occurred.

The MESSAGE\_LEVEL configuration option in each Workload Automation 5 component lets you specify which messages are written (see [Message Levels](#), below).

#### Message ID

Each message is prefixed with a message ID that identifies the message.

The message ID format is pppnnnnl, where:

- ppp is the product category identifier:
  - UAG (Universal Automation Center Components)
  - UNV (Universal Components)
- nnnn is the message number.
- l is the message severity level:
  - A (Audit)
  - I (Informational)
  - W (Warning)
  - E (Error)
  - T (alerT)
  - D (Diagnostic)

#### Message Levels

Each Workload Automation 5 component includes a MESSAGE\_LEVEL configuration option that lets you select which levels (that is, severity levels) of messages are to be written.

<b>Audit</b>	Specifies that all audit, informational, warning, and error messages are to be written.
--------------	---

<b>Informational</b>	Specifies that all informational, warning, and error messages are to be written.
<b>Warning</b>	Specifies that all warning and error messages are to be written.
<b>Error</b>	Specifies that all error messages are to be written.
<b>Trace</b>	Specifies that a trace file is created, to which data used for program analysis will be written. The trace file name and location are component-dependent (see the appropriate Workload Automation 5 component documentation for details). (Trace should be used only at the request of Stonebranch, Inc. Customer Support.)

**Note**

Diagnostic and Alert messages always are written, regardless of the level selected in the MESSAGE\_LEVEL option.

## Message Destinations

The location to which messages are written is the message destination.

Some Workload Automation 5 components have a MESSAGE\_DESTINATION configuration option that specifies the message destination. If a program is used only from the command line or batch job, it may have only one message destination, such as standard error.

Valid message destination values depend on the host operating system.

### z/OS Message Destinations

Workload Automation 5 on z/OS run as batch jobs or started tasks. Batch jobs do not provide the MESSAGE\_DESTINATION option. All messages are written to the **SYSOUT** ddname.

Started task message destinations are listed in the following table.

Destination	Description
LOGFILE	Messages are written to ddname UNVLOG.  All messages written to log files include a date and time stamp and the program's USS process ID.
SYSTEM	Messages are written to the console log as WTO messages.

### Windows Message Destinations

Message destinations are listed in the following table.

Destination	Description
STDERR	Messages are written to standard error. This destination is most useful for console commands.
LOGFILE	Messages are written to a log file. Not all programs provide this destination.  Log files are written to product specific log directories, which can be modified with the LOG_DIRECTORY option. All messages written to log files include a date and time stamp and the program's process ID.
SYSTEM	Messages are written to the Windows Application Event Log.

### UNIX Message Destinations

Message destinations are listed in the following table.

Destination	Description
STDERR	Messages are written to standard error. This destination is most useful for console commands.
LOGFILE	<p>Messages are written to a log file. Not all programs provide this destination.</p> <p>The recommended directory for log files is <code>/var/opt/universal/log</code>. This can be changed with the <code>LOG_DIRECTORY</code> option. All messages written to log files include a date and time stamp and the program's process ID.</p>
SYSTEM	<p>Messages are written to the <b>syslog</b> daemon. Not all programs provide this destination.</p> <p>Universal programs that execute as daemons write to the <b>syslog</b>'s daemon facility. All messages include the programs process ID. If an error occurs writing to the <b>syslog</b>, the message is written to the system console.</p>

### IBM i Message Destinations

Message destinations are listed in the following table.

Destination	Description
STDERR	Messages are written to standard error. A batch job's standard error file is allocated to the print file QPRINT.
LOGFILE	Messages are written to the job's job log.
SYSTEM	Messages are written to the system operator message queue QSYSOPR.

### HP NonStop Message Destinations

Message destinations are listed in the following table.

Destination	Description
STDERR	Messages are written to standard error.
LOGFILE	<p>Messages are written to a log file. Not all programs provide this destination.</p> <p>Log files are written the <b>\$SYSTEM.UNVLOG</b> subvolume. All messages written to log files include a date and time stamp and the program's process ID.</p>

## Auditing

### Auditing

Within Workload Automation 5, an event is the occurrence of some action or condition at a particular location in the computer network and at a particular time at that location. There are a number of different types of events, such as the start of a Workload Automation 5 component, a user authentication failure, or a file transfer completion.

The Universal Event Subsystem (UES) provides the means by which Workload Automation 5 components generate data about those events and, in a single repository, have those events recorded. This collection of recorded events (that is, the event records) is maintained in the UES database and archived to external storage. It represents the work and activity of all distributed workload managed by Workload Automation 5 components.

Workload Automation 5 consists of a set of components distributed across a computer network. The components work together to perform some unit of work. The components that are working together have an association that must be maintained in the event data. For that reason, UES event records not only include information about the event, but also information about associations between the components reporting the events.

The Universal Enterprise Controller (UEC) maintains a central UES database for all event data within its domain of responsibility. The UES database contains all UES event records collected by UEC from Universal Broker components that are defined to it. The UES database provides medium-term persistent storage for the UES events. Periodically, the UES database events must be exported to long-term storage in order to maintain a historical record of events. If the export is not performed periodically, the UES database will continue to grow and eventually exhaust all disk space available to it.

Examples of components and their associations are:

- Universal Command Manager is associated with a remote Universal Command Server, and the Universal Command Server is associated with the job process it has started on behalf of the Universal Command Manager.
- Universal Data Mover Manager is associated with a remote Universal Data Mover Server, and the Universal Data Mover Server is associated with a file being transferred on behalf of the Universal Data Mover Manager.

The components and their associations partly define the Workload Automation 5 architecture. This section provides the necessary understanding of the Workload Automation 5 architecture as presented by the UES event data.



## Creating Write-to-Operator Messages - Examples

### Creating Write-to-Operator Messages Examples

- [Issue WTO Message to z/OS Console](#)
- [Issue WTO Message to z/OS Console and Wait for Reply](#)

## Issue WTO Message to zOS Console

### Issue WTO Message to z/OS Console

The following illustrates the issuing of a WTO message to the z/OS console.

No reply is required.

```
uwto -msg "This message is written to the Console"
```

The message text "**This message is written to the Console**" will be written to the default z/OS consoles.

### SYSIN Options

The SYSIN option used in this example is:

Option	Description
<code>-msg</code>	Text to write to the z/OS operator console. The text is written as a single-line WTO or WTOR message.

### Components

[Universal Write-to-Operator](#)

## Issue WTO Message to zOS Console and Wait for Reply

### Issue WTO Message to zOS Console and Wait for Reply

The following illustrates the issuing of a WTOR message to the z/OS console.

A reply is required.

```
uwto -msg "This message is written to the Console" -reply yes -timeout 120
```

The message text "**This message is written to the Console**" will be written to the default z/OS consoles.

The process will wait 120 seconds for a required reply. If a reply is not received within this time, the WTOR message is deleted and Universal WTO ends with exit code 2. The reply length is limited to 119 characters. The reply is written to UWTO's standard output file.



#### Note

A valid operator reply to a WTOR message can be zero characters. In this case, nothing is written to stdout.

### SYSIN Options

The SYSIN options used in this example are:

Option	Description
-msg	Text to write to the z/OS operator console. The text is written as a single-line WTO or WTOR message.
-reply	Directs Universal WTO to issue a WTOR message and wait for an operator reply to the message.
-timeout	Number of seconds to wait for a WTOR operator reply. If a reply is not received within this time, the WTOR message is deleted and UWTO ends with exit code 2. Default is 0 (wait indefinitely).

### Components

[Universal Write-to-Operator](#)

## **Infitran - Message Translation**

## Message Translation - Overview

- Message Translation
- Usage
  - Translation Table
  - Matching Algorithm

### Message Translation

Indesca component error messages are translated, by the Universal Message Translator utility, into return (exit) codes based on a user-defined translation table.

Every command ends with a return code that indicates the success or failure of the command execution. Typically, a return code of 0 indicates success; all other codes indicate failure.

However, a small number of commands do not set their return code under failure conditions; instead, they issue error messages. Based on the user-defined translation table, Universal Message Translator translates these error messages into return codes.

### Usage

Universal Message Translator requires two input files:

1. Message Input file (user-specified or standard input) containing the error messages that are to be translated into a return codes.
2. Translation Table file containing the user-defined translation table that controls the error message-to-return code translation process.

To perform a translation, Universal Message Translator:

1. Reads the messages in the input file.
2. Matches each line against the translation table entries.
3. Exits with an return code from the best match in the translation table.

If no match is found, Universal Message Translator ends with return code 0.

Universal Message Translator performs operations specified by the configuration options. This section describes each option and their syntax.

### Translation Table

The translation table specifies:

- Text to search for.
- Return code associated with the text.
- Precedence when multiple matches are found.

### Translation Table Format

The translation table consists of one or more lines.

Each line is either:

- Comment line (# in column one)
- Blank line (ignored)
- Translation table entry

Translation table entries consist of two fields separated by spaces or tabs. An entry cannot be continued onto multiple lines.

### Translation Table Fields

Field	Description
Message Mask	Selects which messages to match in the input file. The mask must be enclosed in double ( " ) quotation marks.  Mask characters include the asterisks ( * ) and the question mark ( ? ). The asterisk matches 0 or more characters and the question mark matches one character.

	If an asterisk, question mark, or quotation mark is required in the message text, it must be preceded with a back slash (\). If a back slash is required in the message text, it must be preceded by another back slash.
Exit Code	Specifies an integer value that UMET exits with if this entry is the resulting match.  The exit code is in the range of -99999 to 99999.

## Matching Algorithm

The input file is read line by line. For each line, the line is compared to each entry in the translation table. All the matching entries are saved.

After the entire input file is read, the matched entries from the translation table are sorted in ascending order by their line number in the translation table. The first entry in this sorted list is the resulting translation table entry. The exit code from the resulting translation table entry is used as the return code of UMET. If no matching entry is found, UMET exits with 0.



### IBM i

The resulting return code from the translation process is converted into an IBM i escape message.

The escape message ID and message severity depend on the return code value as identified in the following table.

Return Code	Message ID	Message Severity
1 – 10	UNV0344	10
11 – 20	UNV0345	20
21 – 30	UNV0346	30
31 and higher	UNV0347	40

## Message Translation - Examples

### Message Translation - Examples

- [Translating Error Messages](#)
- [Execute Universal Message Translator from zOS](#)
- [Execute Universal Message Translator from Windows](#)
- [Execute Universal Message Translator from UNIX](#)
- [Execute Universal Message Translator from IBM i](#)

**Note**

The IBM i example references the IBM i command by its untagged name. If you are using commands with tagged names to run [Universal Message Translator](#), substitute the tagged name for this untagged name. (For information on tagged names, see [UCHGRLS \(Change Release Tag\) Program](#).)

## Translating Error Messages

- Example 1
- Example 2
  - Components



### Note

These examples are not specific to any particular operating system.

### Example 1

In this example, a command generates the following **stderr** file.

```
Error opening rc file /etc/arc.rc
No rc file opened.
Ending due to error.
```

From the contents of the message file, we can see that the program failed to open a resource configuration file.

Either of the following translation tables could match error messages in the message file. Message masks should be general enough to match a set of error messages.

#### Translation Table 1

```
# UMET Translation Table 1
#
# Message Mask           Exit Code
# -----               -
# "*error*"              8
```

Translation Table 1 will result in a match if any input line contains the word **error**. The resulting exit code will be *8* if a match occurs.

#### Translation Table 2

```
# UMET Translation Table 2
#
# Message Mask           Exit Code
# -----               -
# "Ending due to error." 8
```

Translation Table 2 will result in a match only if the exact message text **"Ending due to error."** appears as a line in the input file. This is less general, but may be sufficient for this command.

### Example 2

(This example continues from Example 1.)

In this example, the command now generates the following **stderr** file.

```
Error opening rc file /etc/arc.rc
Processing rc file /usr/etc/arc.rc
Ending successfully
```

From the contents of the message file, we can see that the program failed to open a resource configuration file `/etc/arc.rc`, but successfully opened file `/usr/etc/arc.rc`.

#### Translation table



The following translation table is one of many that could match error messages in the message file.

```
# UMET Translation Table 1
#
# Message Mask           Exit Code
# -----
"Ending due to error."   8
"Processing rc file *"   0
"Error opening rc file *" 8
```

Translation Table 1 contains three entries:

- First entry matches against a specific error message that always indicates an error if present.
- Second and third entries match messages produced by resource configuration file processing.

## Components

[Universal Message Translator](#)

## Execute Universal Message Translator from z/OS

- [Execute Universal Message Translator from z/OS](#)
  - [PARM Options](#)
  - [Components](#)

### Execute Universal Message Translator from z/OS

The following figure illustrates the execution of Universal Message Translator from z/OS.

```
//S1 EXEC PGM=UMET,PARM='-table tabledd -level verbose'
//STEPLIB DD DISP=SHR,DSN=hlq.UNV.SUNVLOAD
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//CEDUMP DD SYSOUT=*
//TABLEDD DD *
**ERROR**      8
**WARN**       4
**ERROR**      7
/*
//SYSIN DD *
THIS IS AN ERROR MESSAGE RESULTING IN RETURN CODE 8.
/*
```

The `-table` option points to the DD statement **TABLEDD**, which defines the return codes to end this process based on matching text. The first column defines the text to match; the second defines the return code to set if the matching text exists in the **SYSIN** DD.

The `-level` option turns on messaging. All messages will be written to **SYSPRINT**. The **SYSIN** DD statement points to the text file to be interrogated.

#### PARM Options

The PARM options used in this example are:

Option	Description
<code>-table</code>	Translation table file name.
<code>-level</code>	Level of messages that will be displayed.

#### Components

[Universal Message Translator](#)

## Execute Universal Message Translator from Windows

- [Execute Universal Message Translator from Windows](#)
  - [Command Line Options](#)
  - [Components](#)

### Execute Universal Message Translator from Windows

The following figure illustrates the execution of Universal Message Translator from Windows.

```
-table c:\umettable.txt -file c:\umetfile.txt -level verbose
```

The `-table` option points to the file that defines the return codes with which to end this process, based on matching text.

The first column of the file defines the text to match; the second defines the exit code to set if the matching text exists in the file defined by the `-file` option.

The `-level` option turns on messaging. All messages will be written to **stdout**.

### Command Line Options

The command line options used in this example are:

Option	Description
<code>-table</code>	Translation table file name.
<code>-level</code>	Level of messages that will be displayed.
<code>-file</code>	Input message file name. If the option is not specified, UMET reads its input from <b>stdin</b> .

### Components

[Universal Message Translator](#)

## Execute Universal Message Translator from UNIX

- [Execute Universal Message Translator from UNIX](#)
  - [Command Line Options](#)
  - [Components](#)

### Execute Universal Message Translator from UNIX

The following figure illustrates the execution of Universal Message Translator from UNIX.

Although the command is shown on two lines, it should be entered on one line at the command prompt or within a script, or it can be continued within the script with the UNIX continuation character `\`.

```
/opt/universal/ucmdsrv-2.2.0/bin/umet -table /tmp/umettable.txt -file /tmp/umetfile.txt -level verbose
```

The `-table` option points to the file, which defines the return codes with which to end this process, based on matching text.

The first column of the file defines the text to match; the second defines the return code to set if the matching text exists in the file defined by the `-file` option.

The `-level` option turns on messaging. All messages will be written to **stdout**.

### Command Line Options

The command line options used in this example are:

Option	Description
<code>-table</code>	Translation table file name.
<code>-level</code>	Level of messages that will be displayed.
<code>-file</code>	Input message file name. If the option is not specified, UMET reads its input from <b>stdin</b> .

### Components

[Universal Message Translator](#)

## Execute Universal Message Translator from IBM i

- [Execute Universal Message Translator from IBM i](#)
  - [Command Line Options](#)
  - [Components](#)

### Execute Universal Message Translator from IBM i

The following example illustrates the execution of Universal Message Translator from IBM i.

```
STRUME MSGFILE(input_file) MSGMBR(member) TBL(table_file) TBLMBR(member) MSGLEVEL(*VERBOSE)
```

The `TBL [TBLMBR]` option points to the file, which defines the exit codes with which to end this process, based on matching text. The first column of the file defines the text to match; the second defines the return code to set if the matching text exists in the file defined by the `MSGFILE [MSGMBR]` option.

Diagnostic message UNV0383 and Informational message CPF9815 are issued if an error occurs during execution of the STRUME command. All other informational messages will be written to STDOUT. To avoid messages written to stdout, either allow `MSGLEVEL` to default to `*warn` or specify `MSGLEVEL` as `*error`.

### Command Line Options

The command line options used in this example are:

Option	Description
<code>TBL [TBLMBR]</code>	Translation table file name.
<code>MSGLEVEL</code>	Level of messages that will be displayed.
<code>MSGFILE [MSGMBR]</code>	Input message file name. If the option is not specified, UMET reads its input from <code>stdin</code> , which is allocated to the terminal for interactive jobs and to QINLINE for non-interactive jobs.

### Components

[Universal Message Translator](#)

## **Infitran - Monitoring and Alerting**

## Monitoring and Alerting - Overview

- [Monitoring and Alerting](#)
- [Monitoring of All Agents](#)
  - [Monitored Information](#)
  - [Polling](#)
  - [Alerts](#)
- [Querying for Job Status and Activity](#)

### Monitoring and Alerting

The Monitoring and Alerting feature of Infitran provides for the monitoring the status and activity of all Infitran Agents in an enterprise and the posting of alerts regarding the statuses.

Monitoring is provided through continuous [Monitoring of All Agents](#) or by [Querying for Job Status and Activity](#) of a specific Agent.

### Monitoring of All Agents

Infitran provides for the continuous monitoring of all Agents in an enterprise through its Universal Enterprise Controller component.

#### Monitored Information

Infitran monitors for three types of information:

1. Alerts for all Agents and SAP systems being monitored
2. Jobs (active, completed, and failed) for all Agents being monitored
3. Systems (Agents and SAP systems) being monitored

This information can be viewed via the [I-Activity Monitor](#) client application.

#### Polling

Infitran periodically polls each Agent and SAP system in an enterprise in order to retrieve its status information.

It determines whether or not a change in status of the Agent or SAP system has occurred since the last poll. If the status has changed, it sends this information to the [I-Activity Monitor](#).

#### Alerts

Infitran sends out alerts to any connected Agent-monitoring applications whenever:

- Agent is unreachable.
- Agent is not responding.
- Agent component enters an orphaned or disconnected state.

These alerts are posted to the:

- Event Log (when running under Windows)
- Console (when running under z/OS)

Automation tools can be used in conjunction with these messages to perform operations based on agent failures.

### Querying for Job Status and Activity

Infitran has the ability to query any specific Universal Broker in an enterprise for Broker-related, and active component-related, activity via the [Universal Query](#) utility.

Universal Query returns information for a Universal Broker that is installed on the host, as specified by configuration options on the command line or in a configuration file. Information regarding the components managed by a particular Universal Broker also can be requested.

Universal Query registers with a locally running Universal Broker. Consequentially, a Universal Broker must be running in order for a Universal Query to execute.

## Querying for Job Status and Activity - Examples

### Querying for Job Status and Activity Examples

- [Querying - Universal Query Output](#)
- [Querying - Universal Query for z/OS](#)
- [Querying - Universal Query for UNIX and Windows](#)
- [Querying - Universal Query for IBM i](#)



**Note**

This example references the IBM i command by its untagged name. If you are using commands with tagged names to run [Universal Query](#), substitute the tagged name for this untagged name. (For information on tagged names, see [UCHGRLS \(Change Release Tag\) Program](#).)



## Querying - Universal Query Output

### Universal Query Output

The following figure illustrates an example of the output generated by the execution of the Universal Query utility.

This sample output is from the execution of Universal Query to host `dallas.domain.com` using a NORMAL report.

```

                Universal Query Report
                  for
    Mon 23 May 2011 05:54:00 PM EDT

host: 10.20.30.40  port: 7887  ping: NO  report: NORMAL

    Ubroker Host Name...:
    Ubroker IP Address..: *
    Ubroker Host Port...: 7887
    Ubroker Description.: Universal Broker
    Ubroker Version.....: 5.1.0 Level 0 Release Build 108
    Ubroker Service.....: UNKNOWN
    Ubroker Status.....: Active

Component ID.....: 1121367481
Component Name.....: ucmd
Component Description.....: Universal Command Server
Component Version.....: 5.1.0 Level 0 Release Build 108
Component Type.....: ucmd
Component Process ID.....: 773
Component Start Time.....: 05:53:39 PM
Component Start Date.....: 05/23/2011
Component Command ID.....: sleep 60
Component State.....: REGISTERED
Component MGR UID.....: ucuser
Component MGR Work ID.....: PID12890
Component MGR Host Name...: dallas.domain.com
Component MGR IP Address..: 10.20.30.34
Component MGR Port.....: 49082
Component Comm State.....: ESTABLISHED
Component Comm State Time.: 05:53:41 PM
Component Comm State Date.: 05/23/2011
Component MGR Restartable.: NO
Component Comment.....: Sleep for 60 secs on dallas
```

## Querying - Universal Query for zOS

### Universal Query for z/OS

The Universal Query utility is used to list all active components on a remote server.

The output will be written to the **SYSPRINT** DD statement.

```
//S1 EXEC UQRYPRC
//SYSIN DD *
-host dallas
/*
```

All active component information for server **dallas** will be written to DD statement **SYSOUT**.

### SYSIN Option

The SYSIN option used in this example is:

Option	Description
-host	Directs the command to a computer with a host name of <b>dallas</b> .

### Components

Universal Query

## Querying - Universal Query for UNIX and Windows

### Universal Query for UNIX and Windows

The Universal Query utility is used to list all active components on a remote server.

The output will be written to stdout.

```
uquery -host localhost
```

All active component information for the **localhost** server will be written to stdout.

### Command Line Option

The command line option used in this example is:

Option	Description
<code>-host</code>	Directs the command to the <b>localhost</b> .

### Components

Universal Query

## Querying - Universal Query for IBM i

### Universal Query for IBM i

The Universal Query utility is used to list all active components on a remote server.

The output will be written to stdout.

```
STRUQR HOST(localhost) PORT(4990)
```

This command provides active component information for the **localhost** server listening on port 4990 will be written to stdout.

```
STRUQR HOST(fortworth)
```

This command provides active component information from the **fortworth** server listening on the default port 7887.

### Command Line Options

The command line options used in these examples are:

Option	Description
HOST	Directs the command to the <b>localhost</b> .
PORT	TCP port on the remote server.

### Components

Universal Query

## Infitran - Windows Event Log Dump

## Windows Event Log Dump - Overview

### Windows Event Log Dump

Infitran provides the ability to select records from a Windows event log and write them to a specified output file via its Universal Event Log Dump utility.

All records from a log can be dumped, or event records can be selected according to the date and time that they were generated.

Universal Event Log Dump can be run any time as a stand-alone application. It also is designed to work with Indesca's Universal Command, which provides centralized control from any operating system and additional options for redirecting output.

Universal Event Log Dump consists of the command line program (**ueld**) followed by a list of configuration options.

## Windows Event Log Dump - Examples

### Windows Event Log Dump Examples

- [Execute Universal Event Log Dump from a Windows Server](#)

## Execute Universal Event Log Dump from a Windows Server

### Execute Universal Event Log Dump from a Windows Server

The following figure illustrates the execution of Universal Event Log Dump from a Windows server.

The application log, from the previous day at 15:00 until current time, will be dumped to a file on the server.

```
ueld -logtype APPLICATION -stime "*-1,15:00 PM" -file c:\application.log
```

### Command Line Options

The command line options used in this example are:

Command Options	Description
<a href="#">-logtype</a>	Event log to be dumped.
<a href="#">-stime</a>	Starting date and time.
<a href="#">-file</a>	Complete path to the file that will be used to store the selected event log records.

### Components

[Universal Event Log Dump](#)



# Infitran - Fault Tolerance Implementation

- [Overview](#)
- [Network Fault Tolerance](#)
  - [Open Retry](#)
  - [Component Management](#)
  - [Communication State Values](#)

## Overview

For Infitran, fault tolerance is the capability of its Workload Automation components to recover or restart from an array of error conditions that occur in any large IT organization.

Errors occur as a result of human, software, or hardware conditions. The more resilient a product is to errors, the greater value it offers.

Currently, network fault tolerance is implemented in one Infitran component:

- [Universal Data Mover](#)

## Network Fault Tolerance

UDM uses the TCP/IP protocol for communications over a data network. The TCP/IP protocol is a mature, robust protocol capable of re-sending packets and rerouting packets when network errors occur. However, data networks do have problems significant enough to prevent the TCP/IP protocol from recovering. As a result, the TCP/IP protocol terminates the connection between the application programs. Like any application using TCP/IP, UDM is subject to these network errors. Should they occur, a product can no longer communicate and must shutdown or restart. These types of errors normally show themselves as premature closes, connection resets, time-outs, or broken pipe errors.

UDM provides the ability to circumvent these types of errors with its Network Fault Tolerant protocol. By using the network fault tolerant protocol, UDM traps the connection termination caused by the network error and it reestablishes the network connections. Once connections are reestablished, processing automatically resumes from the location of the last successful message exchange. No program restarts are required and no data are lost.

The network fault tolerant protocol acknowledges and checkpoints successfully received and sent messages, respectively. The network fault tolerant protocol does reduce data throughput. Consequentially, the use of network fault tolerance should be carefully weighed in terms of increased execution time versus the probability of network errors and cost of such errors. For example, it may be easier to restart a program then to incur increased execution time.

When a network connection terminates, the UDM Manager will enter a network reconnect phase. In the reconnect phase, the Manager attempts to connect to the UDM Server and reestablish its network connections. The condition that caused the network error may persist for only seconds or days. The Manager will attempt Server reconnection for a limited amount of time (configured with the [RECONNECT\\_RETRY\\_COUNT](#) and [RECONNECT\\_RETRY\\_INTERVAL](#) options). These two options determine, respectively, how many reconnect attempts are made and how often they are made. After all attempts have failed, the manager ends with an error.

When a network connection terminates, the Server enters a disconnected state and waits for the Manager to reconnect. The user process continues running; however, if the user process attempts any I/O on the standard files, it will block. The Server waits for the Manager to reconnect for a period of time defined by the Manager's [RECONNECT\\_RETRY\\_COUNT](#) and [RECONNECT\\_RETRY\\_INTERVAL](#). Once that time has expired, the Server terminates the user process and exits.

UDM can request the use of the network fault tolerant protocol. If the Server does not support the protocol or is not configured to accept the protocol, the Manager continues without using the protocol.

The [NETWORK\\_FAULT\\_TOLERANT](#) and [RECONNECT\\_RETRY\\_INTERVAL](#) option is used to request the protocol.

## Open Retry

Open Retry is a type of fault tolerance used at the session-establishment level.

(Network fault tolerance is used from the time that a session has been fully established until the session has terminated.)

Open Retry is used during the establishment phase of a session. UDM tries to establish a session when the [open](#) command is issued. If the [OPEN\\_RETRY](#) option value is **yes**, and UDM fails to establish the session due to a network error, timeout, or the inability to start a transfer server, it will retry the open command based on the settings of the [OPEN\\_RETRY\\_COUNT](#) and [OPEN\\_RETRY\\_INTERVAL](#) options.

## Component Management

In order to fully understand Universal Data Mover fault tolerant features, some understanding of how the Universal Broker manages components is necessary.

Universal Broker manages component startup, execution, and termination. The broker and its components have the ability to communicate service requests and status information between each other.


The Broker maintains a database of components that are active or have completed and waiting for restart or reconnection. The component information maintained by the broker determines the current state of the component. This state information is required by the broker to determine if a restart or reconnect request from a manager is acceptable or not. The broker's component information can be viewed with the Universal Query program.

One piece of component information maintained by the broker is the component's communication state. The communication state primarily determines what state the Universal Data Mover Server is in regarding its network connection with a manager and the completion of the user process and its associated spooled data.

## Communication State Values

The following table describes the communication state values.

- **Reconnect** column indicates whether or not a network reconnect request is valid.
- **Restart** column indicates whether or not a restart request is valid.

State	Reconnect	Restart	Description
COMPLETED	NO	NO	Server and manager have completed. All standard output and standard error files have been sent to the manager and the user process's exit status.
DISCONNECTED	YES	YES	<p>Server is not connected to the manager. This occurs when a network error has occurred, the manager halted, or the manager host halted.</p> <p>The server is executing with either the network fault tolerant protocol, is restartable, or both.</p> <div style="background-color: #ffffcc; padding: 5px; margin-top: 10px;"> <p> <b>Note</b> The server cannot tell if the manager is still executing or not since it cannot communicate with it.</p> </div>
ESTABLISHED	NO	NO	Server and manager are connected and processing normally. This is the most common state when all is well.
RECONNECTING	NO	NO	<p>Server has received a reconnect request from the manager to recover a lost network connection.</p> <p>This state should not remain long, only for the time it takes to re-establish the network connections.</p>
STARTED	NO	NO	<p>Server has started.</p> <p>If the server is restartable it is receiving the standard input file from the manager and spooling it.</p>



## **Infitran - Network Data Transmission**

## Network Data Transmission - Overview

### Infitran - Network Data Transmission

Distributed systems, such as Workload Automation 5, communicate over data networks. All Infitran components communicate using the TCP/IP protocol. The UDP protocol is not used for any product data communication over a network.

#### Network Protocols

Infitran can utilize either of two network protocols:

1. [SSL \(Secure Socket Layer\) Protocol](#)

Secure Socket Layer version 3 (**SSLv3**) provides the highest level of security available. SSL 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.

All Workload Automation 5 components (version 3.x and later) use **SSLv3**.

2. [Universal V2 Protocol](#)

Universal V2 (version 2) legacy protocol, **UNVv2**, is provided for backward compatibility with Workload Automation (formerly Universal Products) versions earlier than 3.x, and when the SSL protocol resource utilization is considered too high.

To ensure backward compatibility, this protocol is still supported by version 3.x components.

In addition to the network protocol used to transmit data, [Universal Application Protocol](#) is discussed as well.

## SSL (Secure Socket Layer) Protocol

- [Overview](#)
- [Data Privacy and Integrity](#)
  - [Encryption Algorithms](#)
  - [Message Digest Algorithms](#)
  - [Supported SSL Cipher Suites](#)
- [Peer Authentication](#)

### Overview

Workload Automation implements the SSL protocol using the OpenSSL library or the IBM z/OS System SSL library, available on the z/OS operating system. 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 Workload Automation 5. **TLSv1** is more commonly referred to simply as SSL and the term SSL is used throughout the rest of this documentation to mean **TLSv1**, unless otherwise noted.

The SSL protocol addresses the major challenges of communicating securely over a potentially insecure data network. This page discusses the issue of data privacy and integrity, and peer authentication.

### Data Privacy and Integrity

People with sufficient technical knowledge and access to network resources can watch or capture data transmitting across the network. What they do with the data is up to them.

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

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. Additionally, although data may be encrypted, there is nothing stopping a malicious person from changing the data 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. It may be accepted as valid data, but the information it represents is now erroneous.

Data integrity must be protected from errors in transmission and malicious users. Data integrity checks insure that what was sent is exactly what is received by the recipient. Without integrity checks, there is no guarantee.

### Encryption Algorithms

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.

### Message Digest Algorithms

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 maintained, else the data is rejected as it has been modified.

Message digest algorithms are often referred to as MACs and can be used synonymously in most contexts.

### Supported SSL Cipher Suites

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

Workload Automation 5 supports a subset of the complete SSL cipher suites defined by the standard. The cipher suite name is formatted as an encryption algorithm abbreviation followed by the message digest algorithm abbreviation.

The following table identifies the supported SSL 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

Workload Automation 5 supports one additional cipher suite name that is not part of the SSL protocol. The NULL-NULL cipher suite turns SSL off completely and instead uses the [Workload Automation Protocol \(UNVv2\)](#).

### Selecting an SSL Cipher Suite

When two Workload Automation 5 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 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 Server supports. The first cipher suite in common is the one used for the session.

Lists of cipher suites are helpful where 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 Manager has both cipher suites in its list, it can negotiate either cipher suite with servers of different security levels.

## Peer Authentication

When communicating with a party across a data network, how do you insure that the party you are communicating with (your peer) is who you believe? A common form of network attack is a malicious user representing themselves as another user or host.

Peer authentication insures that the peer is truly who they identify themselves as. Peer authentication applies to users, computer programs and hardware systems.

SSL uses X.509 certificates and public and private keys to identify an entity. An entity may be a person, a program, or a system. A complete description of X.509 certificates is beyond the scope of this documentation. [X.509 Certificates](#) provides an overview to help get the reader oriented to the concepts, terminology and benefits.

For additional details, the following web site is recommended:

<http://www.faqs.org/rfcs/rfc3280.html>

## Universal V2 Protocol

### Universal V2 Protocol

The Universal V2 protocol, **UNVv2**, is a proprietary protocol that securely and efficiently transports data across data networks. **UNVv2** was the only supported protocol in Workload Automation (formerly Universal Products) prior to version 3 and will be available in future versions.

**UNVv2** addresses data privacy and integrity. It does not address peer authentication.

### Data Privacy and Integrity

Data privacy is insured with data encryption algorithms. **UNVv2** utilizes 128-bit RC4 encryption for all data encryption.

Data integrity is insured with message digest algorithms. **UNVv2** utilizes 128-bit MD5 MACs for data integrity. **UNVv2** referred to data integrity as data authentication.

Encryption and integrity may be enabled and disabled on an individual bases.

Encryption keys are generated using a proprietary key agreement algorithm. A new key is created for each and every network session.



## Universal Application Protocol

- Universal Application Protocol
- Low-Overhead
- Secure
- Extensible

## Universal Application Protocol

Indesca components use an application-layer protocol to exchange data messages. The protocol has the following characteristics:

- Low-Overhead
- Secure
- Extensible
- Configurable Options

The following information refers to two categories of data transmitted by Workload Automation:

1. Control data (or messages) consists of messages generated by Workload Automation components in order to communicate with each other. The user of the product has no access to the control data itself.
2. Application data (or messages) consists of data that is transmitted as part of the requested work being executed. For example, standard input and output data of jobs Universal Command executes. The data is created by the job and read or written by Universal Command on behalf of the job.

## Low-Overhead

The protocol is lightweight, in order to minimize its use of network bandwidth. The product provides application data compression options, which reduces the amount of network data even further.

There are two possible compression methods:

1. **ZLIB** method offers the highest compression ratios with highest CPU utilization.
2. **HASP** method offers the lowest compression ratios with lowest CPU utilization.



### Note

Control data is not compressed. Compression options are available for application data only.

## Secure

When used by Workload Automation 5 Managers prior to version 3.x, and when communicating with Workload Automation 5 Servers that force encryption on, the UNVv2 protocol is secure.

All control data exchanged between Workload Automation 5 components are encrypted with a unique session key and contain a MAC. The encryption prevents anyone from analyzing the message data and attempting to circumvent product and customer policies. Each session uses a different encryption key to prevent "play back" types of network attacks, where messages captured from a previous session are replayed in a new session. This applies to both network protocols: SSL and UNVv2.

In versions prior to Workload Automation 5, the security features used in the control messages are not optional. They cannot be turned off. The security features are optional for application data sent over the network.

Starting with Workload Automation 5, the UNVv2 protocol is used only when SSL is disabled on the control session by specifying the NULL-NUL cipher suite. In this case, the UNVv2 encryption or MACs are not used for control messages.

As of Workload Automation 5, the SSL protocol must be used if data privacy and integrity is required for control messages. For this reason, UNVv2 should only be used when the resource utilization of SSL is considered too high and data privacy is not required. It is Stonebranch's recommendation that SSL should be used if at all possible to insure data privacy and data integrity.

Backward compatibility is still maintained with Workload Automation 5 (formerly Universal Products) versions prior to 3.x such that encryption and MACs are still utilized for the control session.

## Extensible

The message protocol used between the Workload Automation 5 components is extensible. New message fields can be added with each new

release without creating product component incompatibilities. This permits different component versions to communication with each other with no problems. This is a very important feature for distributed systems, since it is near impossible to upgrade hundreds of servers simultaneously.

New encryption and compression algorithms can be added in future releases without losing backward compatibility with older releases. After a network connection is made, connection options are negotiated between the two Workload Automation 5 programs. The options negotiated include which encryption and compression algorithms are used for the session. Only algorithms that both programs implement are chosen in the negotiation process. The negotiation process permits two different program versions to communicate.

## Network Data Transmission - Configurable Options

- Configurable Options
  - CODE\_PAGE
  - CTL\_SSL\_CIPHER\_LIST
  - DATA\_AUTHENTICATION
  - DATA\_COMPRESSION
  - DATA\_ENCRYPTION
  - DATA\_SSL\_CIPHER\_LIST
  - DEFAULT\_CIPHER
  - ENCRYPT\_CONTROL\_SESSION
  - KEEPALIVE\_INTERVAL
  - NETWORK\_DELAY
  - SIO\_MODE

### Configurable Options

The network protocol can be configured in ways that affect compression, encryption, code pages, and network delays.

The following configuration options are available on many Workload Automation components:

#### CODE\_PAGE

The CODE\_PAGE option specifies the code page translation table used to translate network data from and to the local code page for the system on which the program is executing.

A codepage table is text file that contain a two-column table. The table maps local single byte character codes to two-byte UNICODE character codes.

Code pages are located in the product National Language Support (NLS) directory or library. New code pages may be created and added to the NLS directory or library. The CODE\_PAGE option value is simply the name of the code page file without any file name extension if present.

#### CTL\_SSL\_CIPHER\_LIST

The CTL\_SSL\_CIPHER\_LIST option specifies one or more SSL cipher suites that are acceptable to use for network communications on the control session, which is used for component internal communication.

The SSL protocol uses cipher suites to specify the combination of encryption and message digest algorithms used for a session. An ordered list of acceptable cipher suites can be specified in a most-to-least order of preference.

An example cipher suite list is RC4-MD5,RC4-SHA,AES128-SHA. The RC4-MD5 cipher suite is the most preferred and AES128-SHA is the least preferred.

When two Workload Automation components (Manager and a Server) first connect, they perform an SSL handshake that negotiates the cipher suite to use for the session. The 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 Server supports. The first cipher suite in common is the one used for the session.

#### DATA\_AUTHENTICATION

The DATA\_AUTHENTICATION option specifies whether or not the network data is authenticated. Data authentication verifies that the data did not change from the point it was sent to the point it was received.

Data authentication also is referred to as a data integrity in this document.

Data authentication occurs for each message sent over the network. If a message fails authentication, the network session is terminated and both programs end with an error.

The DATA\_AUTHENTICATION option is applicable to the **UNVv2** protocol only. SSL always performs authentication.

#### DATA\_COMPRESSION

The DATA\_COMPRESSION option specifies that network data be compressed.

Compression attempts to reduce the amount of data to a form that can be decompressed to its original form. The compression ratio is the original size divided by the compressed size. The compression ratio value will depend on the type of data. Some data compress better than others.

Two methods of compression are available:

1. ZLIB method provides the highest compression ratio with the highest use of CPU
2. HASP method provides the lowest compression ratio with the lowest use of CPU.

Whether or not compression is used and which compression method is used depends on several items:

- Network bandwidth. If network bandwidth is small, compression may be worth the cost in CPU.
- CPU resources. If CPU is limited, the CPU cost may not be worth the reduced bandwidth usage.
- Data compression ratio. If the data does not compress well, it is probably not worth CPU cost. If the data ratio is high, the CPU cost may be worth it.

## DATA\_ENCRYPTION

The DATA\_ENCRYPTION option specifies whether or not network data is encrypted.

Encryption translates data into a format that prevents the original data from being determined. Decryption translates encrypted data back into its original form.

The type of encryption performed depends on the network protocol being used, SSL or **UNVv2**.

Data encryption does increase CPU usage. Whether or not encryption is used depends on the sensitivity of the data and the security of the two host systems and the data network between the hosts.

## DATA\_SSL\_CIPHER\_LIST

The DATA\_SSL\_CIPHER\_LIST option specifies one or more SSL cipher suites that are acceptable to use for network communications on the data session, which is used for standard I/O file transmission.

(See [CTL\\_SSL\\_CIPHER\\_LIST](#).)

## DEFAULT\_CIPHER

The DEFAULT\_CIPHER option specifies the SSL cipher suite to use (since SSL protocol requires a cipher suite) if the DATA\_ENCRYPTION option is set to **no**. The default DEFAULT\_CIPHER is NULL-MD5 (no encryption, MD5 message digest).

All SSL cipher suites have a message digest for good reasons. The message digest ensures that the data sent are the data received. Without a message digest, it is possible for bits of the data packet to get changed without being noticed.

## ENCRYPT\_CONTROL\_SESSION

The ENCRYPT\_CONTROL\_SESSION option is a server-only option that enforces encryption on the control session. When the option is set to a value of **no**, the server will accept a control session protocol without encryption and message authentication codes (MACs). The default is **yes**.

Starting with Workload Automation 5, a manager can request that the UNVv2 protocol be used without encryption or MACs. Considering that host systems may require differing security policies, this option allows for each server to be configured appropriately based on its security policy.

## KEEPALIVE\_INTERVAL

The KEEPALIVE\_INTERVAL option specifies how often, in seconds, a keepalive message (also commonly known as a heartbeat message) is sent between a manager and server.

A keepalive message ensures that the network and both programs are operating normally. Without a keepalive message, error conditions can arise that place one or both programs in an infinite wait.

A keepalive message is sent from the server to the manager. If the server does not receive a keepalive acknowledgement from the manager in a certain period of time (calculated as the maximum of 2 x [NETWORK\\_DELAY](#) or the KEEPALIVE\_INTERVAL), the server considers the manager or network as unusable.

How the server processes a keepalive time-out depends on what fault tolerant features are being used. If no fault tolerant features are being used, the server ends with an error. The manager expects to receive a keepalive message in a certain period of time (calculated as the KEEPALIVE\_INTERVAL + 2 x [NETWORK\\_DELAY](#)).

## NETWORK\_DELAY

The NETWORK\_DELAY option provides the ability to fine tune Workload Automation 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, Workload Automation components time out waiting for a packet to arrive in a specified period of time. The delay

option specifies this period of time.

NETWORK\_DELAY specifies the maximum acceptable delay in transmitting data between two programs. Should a data transmission take longer than the specified delay, the operation ends with a time out error. Workload Automation components will consider a time out error as a network fault.

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.

## **SIO\_MODE**

The SIO\_MODE option specifies whether the data transmitted over the network is processed as text data or binary data.

Text data is translated between the remote and local code pages. Additionally, end of line representations are converted

Text translation operates in two modes: direct and UCS. The default is direct. The direct translation mode exchanges code pages between Workload Automation components to build direct translation tables. Direct translation is the fastest translation method when a significant amount (greater than 10K) of text data is transmitted. The code page exchange increases the amount of data sent over the network as part of the network connection negotiation.

UCS translation does not require the exchange of code pages. For transactions that have little text data transmission, this is the fastest.

Binary data is transmitted without any data translation.

## **Infitran - zOS CANCEL Command Support**

## **zOS CANCEL Command Support - Overview**

### **zOS CANCEL Command Support**

Infitran network fault tolerance provides users with the ability to execute jobs that will continue to run when the network is down (see [Fault Tolerance Implementation](#)).

However, there are scenarios in which the user may want to cancel an executing job that supports network fault tolerance and have both the manager and server processes terminate immediately. Because of fault tolerance, when the manager is terminated, the server side would begin a connection reestablishment protocol and continue to execute. This would allow the started user job to continue running.

In particular, z/OS supports a CANCEL command that will terminate a job executing on the z/OS operating system.

## zOS CANCEL Command Support - Universal Data Mover

- [Overview](#)
- [Exit Codes](#)
- [Security Token](#)

### Overview

When a Universal Data Mover job is cancelled via the z/OS CANCEL command, the job terminates with either of these exit codes:

- Exit code S122, if it is cancelled with a dump.
- Exit code S222, if it is cancelled without a dump.

Part of the responsibility of a Universal Broker executing on a particular host is to monitor the status of all locally running manager processes on that machine. So, when instructed, that Universal Broker could issue a STOP command to the Universal Data Mover Server process associated with the stopped / ended manager process.

In the case of a Universal Data Mover three-party transfer, both the primary and secondary servers need to be cancelled. The Universal Broker running locally with the cancelled Universal Data Mover Manager process will send a STOP command to the primary server. This primary server will, in turn, forward the STOP command to the secondary server, thus cancelling both servers of the three-party transfer.

### Exit Codes

Through the use of the [SERVER\\_STOP\\_CONDITIONS](#) configuration option, the Universal Data Mover Manager process notifies the locally running Universal Broker of the exit codes that should cause it to terminate the running Server process. With this option, the user can specify a list of exit codes that should trigger the locally running Universal Broker to issue the STOP command to the manager's Universal Data Mover server-side process.

[SERVER\\_STOP\\_CONDITIONS](#) can specify a single exit code or a comma-separated list of exit codes. These stop conditions are passed from the manager to the locally running Universal Broker, which store this and other component-specific data about the executing manager component. When this executing Universal Data Mover Manager process is cancelled or stopped, the locally running Universal Broker detects the ending of the manager process and retrieves its process completion information, which includes the exit code of the manager.

The Universal Broker then compares this exit code with the list of exit codes provided by [SERVER\\_STOP\\_CONDITIONS](#). If a match is found, and network fault tolerance is enabled, the Universal Broker will execute a uctl command to STOP the running Universal Data Mover Server component.

### Security Token

For security purposes, Workload Automation pass around a security token that is used by the locally running Universal Broker to STOP associated Universal Data Mover Server process.

This security token is generated on a component-by-component basis by the Universal Broker process that starts the Universal Data Mover Server. Upon generation, this token is returned to the Universal Data Mover Manager which, in turn, updates its locally running Universal Broker with this token. The locally running Universal Broker then uses this token with the issued STOP command to cancel the running Universal Data Mover Server process.

When this token is received by the Universal Broker processes with the request to STOP the server component, the Broker authenticates the received token with the stored token for the running Universal Data Mover Server process. When the token is authenticated, the Universal Data Mover Server process is STOPPED.



# Infitran - Glossary

## Glossary

This glossary defines terms used within the Infitran business solution:

### Agent

A single Workload Automation installation comprised of one Universal Broker, one Universal Automation Center Agent, and one or more other Workload Automation 5 components, such as Universal Data Mover.)

### API

API (Application Programming Interface) is a set of functions, procedures, methods, classes, or protocols that an operating system, library, or service provides to support requests made by computer programs.

### Asynchronous Communication

Transmission of data or sending of messages without the need to wait for a reply from the destination before continuing with the next operation.

### Automation Center

Automation Center is the Workload Automation 5 solution that provides for scheduling of Indesca and Infitran workload on Workload Automation Agents deployed throughout the enterprise.

### CA

CA (Certification Authority) is a trusted third-party organization that issues digital certificates used to create digital signatures and public-private key pairs, guaranteeing that the individual granted the unique certificate is, in fact, who he or she claims to be.

### Channel

Medium used to convey information from sender to receiver.

### Communications Protocol

Set of standard rules for data representation, signaling, authentication, and error detection required to send information over a communications channel.

### Connector

Component used to allow one system or application to communicate with another system or application. A connector can be embedded within an application or operate as a stand-alone component.

### Container

Application environment that provides a runtime environment that offers services such as security, authentication, transaction management, and deployment to an application developer, thus enabling a faster implementation and rollout.

### EAI tools

EAI (Enterprise Application Integration) tools are used for the unrestricted sharing of data and business processes throughout the networked applications or data sources in an organization.

### GLBA

GLBA (Gramm-Leach-Bliley Act) is a law enabling the consolidation of commercial banks, investment banks, securities firms and insurance companies.

### HIPAA

HIPPA (Health Insurance Portability and Accountability Act) is a law that serves to protect health insurance coverage for workers and their families when they change or lose their jobs.

## **HTTP**

HTTP (HyperText Transfer Protocol), a synchronous request / reply protocol, is the underlying protocol used by the World Wide Web to define how messages are formatted and transmitted, and what actions web servers and browsers should take in response to various commands.

## **Internet Application**

A web application (webapp) that is accessed via a web browser over the Internet.

## **Internet Workload**

Internet workload is any application, service, or function that operates in an Internet environment, such as web applications or container applications, and supports an Internet-based communication protocol such as HTTP or SOAP.

## **JMS**

JMS (Java Message Service) is an API that provides a standard way for Java programs to access and interact with an enterprise asynchronous messaging system. JMS uses both point-to-point (queue-based) and publish / subscribe (topic-based) messaging patterns.

## **JMS Connector**

Component that allows the sending and receiving of JMS messages between applications.

## **Light-Weight Container Architecture (LWCA)**

This architecture, combined with the Federated architecture of the current Workload Automation line, provide your enterprise with a loosely coupled, scalable, and secure solution to your enterprise workload management tasks.

## **Listen MEP**

Listen MEP (Message Exchange Pattern) refers to a component that listens for a message from an application or service.

## **Managed File Transfer**

Software solutions that facilitate the secure transfer of data from one computer to another through a network, such as the Internet, while offering a higher level of security and control than FTP.

## **Managers**

Infitran component that provides client services initiating requests on behalf of the user (for example, a Universal Command manager batch job requesting the execution of a command on a remote server).

## **Message**

Abstract format, or container, for sending data between applications or services. No implementation is implied.

## **Message-Based Application**

A message-based application accesses a target application by sending a message to a queue that is controlled by the target application. This queue must be known and accessible to the application sending the message.

## **Message-Based Workload**

Any application, service, or function that supports a message-based communication protocol such as JMS or MQ.

## **Message Exchange Pattern**

A Message Exchange Pattern (MEP) describes the pattern of messages required by a communications protocol in order to establish or use a communication channel.

## **Message-based application**

Application accessed via a web browser over the Internet or Intranet.

## **MQ Connector**

MQ connector supports workload execution via the MQ messaging protocol using synchronous and asynchronous communication.

## **PKI**

PKI (Public Key Infrastructure) a system of digital certificates, CAs, and other registration authorities that verify and authenticate the validity of each party involved in an Internet transaction.

## **Proxy Certificates**

Proxy certificate is a certificate that is derived from, and signed by, a normal X.509 Public Key End Entity Certificate or by another proxy certificate for the purpose of providing restricted proxying and delegation within a PKI-based authentication system.

## **Publish MEP**

Publish Message Exchange Pattern, or MEP, represents an asynchronous outbound workload execution event that sends a message from an application or service to a target destination. This means that you can request execution of a workload using the JMS protocol to a target JMS provider.

## **Remote Procedure Call**

Remote Procedure Call (RPC) is the most common messaging pattern in SOAP. In RPC, one network node (the client) sends a request message to another node (the server). The server immediately sends a response message to the client. This type of transaction also is known as "request / reply."

## **Request / Reply MEP**

Request / Reply MEP represents an outbound request to a target workload followed by an inbound reply from a target workload. This is a synchronous operation, as the calling party waits, or blocks, for the reply to come back before releasing its resources and moving on to the next task.

## **SAP**

SAP ("System Analysis and Programming Development") is a corporation providing enterprise software applications and support to businesses. SAP ERP is its enterprise resource planning software for managing information and among all company functions.

## **Servers**

Infitran component initiated either by a client or the Universal Broker. All servers are started by the Universal Broker. A manager can request that the Broker initiates a server on its behalf, and the manager and server then work together to perform a service, or a server can be started automatically by the Broker when the Broker starts and stopped when the Broker stops.

## **SOA**

SOA (Service-Oriented Architecture) provides methods for systems development and integration where systems group functionality around business processes, and package these as interoperable services.

SOA also describes IT infrastructure, which allows different applications to exchange data with one another as they participate in business processes.

## **SOAP**

SOAP (Simple Object Access Protocol) is a lightweight XML-based messaging protocol used to encode the information in web service request and response messages before sending them over a network. SOAP messages can be transported using a variety of Internet protocols.

SOAP is used predominantly to provide an interface to web service-based workload or legacy workload with a web service interface.

## **SOAP Connector**

Component that allows the sending and receiving of SOAP messages.

## **SOX**

SOX (Sarbanes-Oxley Act) is a law enacted to ensure accurate financial reporting by public companies.

## **SSL encryption**

SSL encryption uses SSL (Secure Sockets Layer) protocol to encrypt private documents for transmission via the Internet. SSL uses two keys to

encrypt data - a public key known to everyone and a private key known only to the recipient of the message.

## **STDIN**

STDIN (standard in), STDOUT (standard out), and STDERR (standard error) are the standard data streams between a computer program and its environment.

## **xWeb Services Description Language (WSDL)**

WSDL is an XML-based language that provides a model for describing Web services. WSDL is often used in combination with SOAP and XML Schema to provide web services over the Internet.

## **WebSphere XD (Extended Deployment) Environment**

WebSphere is designed to set up, operate, and integrate e-business applications across multiple computing platforms using Java-based Web technologies.

## **Workload**

Jobs, processes, applications, and services that require execution, usually in a scheduler or automation-based environment.

## **X.509 certificates**

Digital certificate issued by a CA that is defined according to the X.509 standard for defining digital certificates.

## **XBP 3.0**

XBP (eXternal Background Processing) 3.0 is the primary SAP interface used by Workload Automation.

## **XD Connector**

XD Connector supports workload execution within the WebSphere Extended Deployment environment using synchronous communication via the SOAP protocol.