Indesca 5.1.0

User Guide

Indesca 5.1.0 User Guide

Indesca - Overview

What is Indesca?

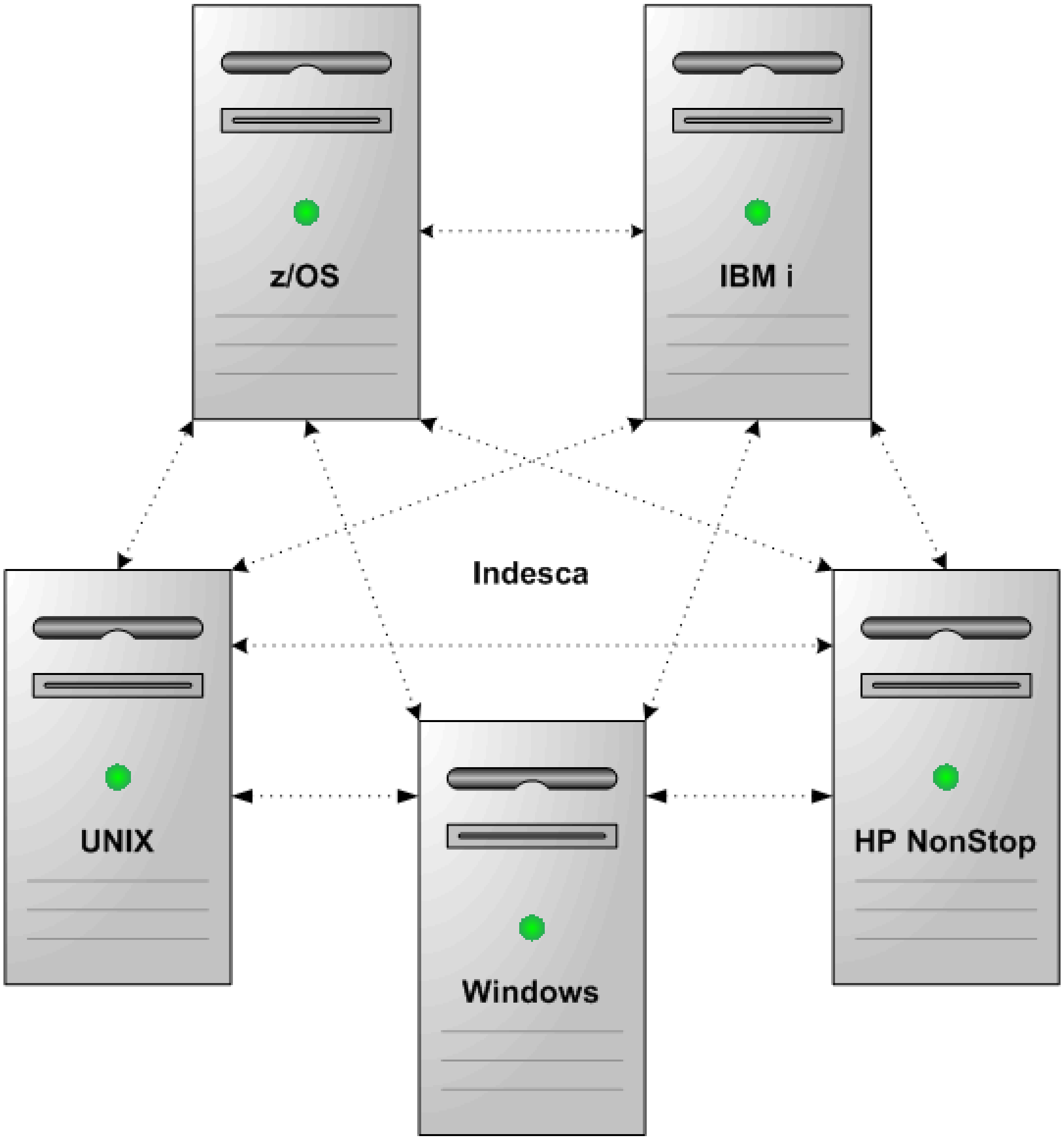
Indesca (Independent Scheduling Agents) is the Stonebranch Inc. business solution for job scheduling.

Indesca integrates with your current scheduling engine, enabling standardized system-wide processes and procedures. It allows job execution without regard to the platform or scheduling solution. It also enables the integration of multiple scheduling solutions. You can set up standardized Indesca processes to execute any workload anywhere in your environment, allowing job scheduling across platforms without specialized platform-dependent scheduling solutions or training.

The single scheduling tool environment allows for centralized monitoring and control over the environment with your existing tools. This allows for proactive management where jobs can be automatically delayed when resources are not available, avoiding time-consuming cleanup after multiple abends.

Indesca allows for integrated support and configuration for new types of workload applications, such as Internet and message-based processing. At the same time, it reduces the complexity of the environment while providing proactive intervention for system maintenance and server failures.

Additionally, Indesca promotes standardization of security policies and central configuration of its components. Other considerations include ease of platform deployment and consolidated audit history.



What Can Indesca Do for Me?

Businesses learn more ways every day to leverage technology for a competitive advantage.

The Information Technology (IT) infrastructure consists of a diverse array of software and hardware systems. Database management, transaction management, resource planning, information warehouse, customer support, e-mail, web servers, and much more are required to sustain a business's technological advantage.

This array of corporate software runs on a large variety of hardware platforms, which in turn run a variety of operating systems. The management of such technology grows more complex each year, if not each month.

The methods, processes, and personnel used to manage the computing environment are as much a part of the business's technology investment as is the software and hardware being managed. Replacing or altering these proven management techniques and tools can be costly as well as risky to a business's success.

Indesca leverages the management resources of today to manage the technology of tomorrow. For example, the z/OS computing environment has been centered around the batch process for years, and for good reason. Nothing else has proven itself to be more easily and reliably managed.

Indesca permits the management of distributed platforms, such as UNIX and Windows, using the same reliable z/OS batch process. The batch processes used to forecast, schedule, manage output, and manage archives can be used to manage the distributed platforms in the same manner.

Indesca Features

The features that make Indesca an independent scheduling agent solution encompass a variety of core and supporting functionality.

The following text describes these features and provides links to detailed information about each one. This includes examples that illustrate feature implementation and links to detailed technical information about the Indesca Components used in that implementation.

The core feature of Indesca is its command line interface that allows Remote Execution of all job scheduling to be initiated – regardless of operating system – *from* any machine in your enterprise *to* any machine in your enterprise.

Similarly, Remote Execution for SAP Systems offers a command line interface that allows you to control background processing tasks in an SAP system from any machine in your enterpriser.

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 Indesca to extend its remote execution functionality to Internet and message-based workload and create file-based events from inbound Internet and message-based application messages.

For Indesca 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.

Indesca also provides a command line interface for Copying Files to and from Remote Systems, whether from manager to server or server to manager.

Indesca’s array of Databases record information throughout an enterprise. Information on all Indesca 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 Indesca system occurrences (events), the action to implement for those events, and the progress of each event.

The Monitoring and Alerting feature of Indesca provides for monitoring the status and activity of all Indesca 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 Indesca’s Universal Configuration Manager graphical user interface.

Additionally, Indesca offers various methods for the Configuration Refresh of all component data. Indesca 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.

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

Fault Tolerance Implementation allows Indesca to recover from an array of error conditions, at both network and component levels, such as may occur in any large enterprise. Since network fault tolerance enables servers to continue processing even after a job is canceled, Indesca’s zOS Cancel Command Support allows – on z/OS operating systems – termination of those jobs.

Indesca Components

Indesca Features are implemented via a set of inter-related components that provide for a complete independent scheduling agent business solution.

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

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.

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.

Universal Event Monitor

Universal Event Monitor Output 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 Indesca 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 Indesca 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 Indesca infrastructure:

I-Activity Monitor

I-Activity Monitor provides end-to-end visibility of workload management activity throughout your Indesca 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 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.

Universal Broker

Universal Broker, required on all systems running Indesca, manages Indesca 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 Indesca business solution, perform a variety of functions for one or more operating systems.

Universal Certificate

Indesca 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 Indesca, 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 Indesca components, and to refresh component configuration data.

Universal Copy

Universal Copy provides a means to copy files from either manager-to-server or server-to-manager.

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 Indesca 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.

Indesca - Remote Execution

Remote Execution - Overview

Overview

This page provides information on the Remote Execution feature of Indesca.

Remote Execution simply refers to the ability of initiating work from one system, referred to as the local system, that executes on another system, referred to as 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 of Indesca is used to execute work on the remote system.

Remote Execution Components

Remote Execution utilizes primarily two Indesca Universal Command (UCMD) components:

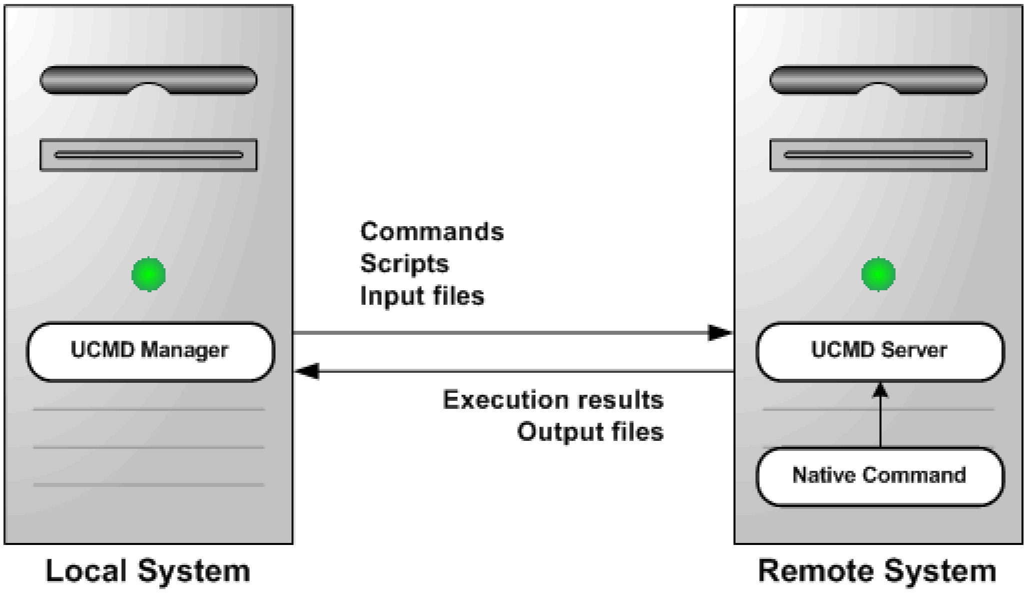
1. Universal Command Manager runs on the local system. The Manager initiates the work on the remote system.
2. Universal Command Server runs on the remote systems. It executes work on behalf of a Universal Command Manager.

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.

Any type of program, command, or script file that can be run from the command line interface can be run by Universal Command. As such, Universal Command interfaces with your platform-specific job scheduling solutions, providing visibility and control throughout your entire enterprise. This enables you to have an end-to-end view of all workload activity.



Remote Execution Examples

The Remote Execution examples illustrated in these pages are specific to the operating systems supported by Workload Automation 5 for the Remote Execution feature of Indesca.

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

⚠ **Note**
In order to keep the 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.

Remote Execution - Primer

Overview

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

Prior to reading this page, read the Overview of Remote execution, as this page 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.

The primer examples assume the Indesca product is installed with default configuration values to help keep the examples consistent and clear. Indesca 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 Universal Command Manager component. All examples use the same set of options (identified in the following table). The actual option names can be different, depending on the operating system on which the UCMD Manager executes. This difference is due to operating system conventions or standards that UCMD abides by.

Remote Execution Primer Examples

The following table describes each of the Universal Command Manager configuration options used in the following primer examples.

Option Name	Description
COMMAND	Command to be executed on the remote system. The command used in the examples is the Windows DOS command 'dir \'. If the remote system is a UNIX system, change the command value to "ls /". If the remote system is an IBM i system, change the command to "DSPLIB QGPL".
REMOTE_HOST	Host name or IP address of the remote system on which the command is to be executed. The examples use a host name of **dallas**. To execute the examples in your environment, change the host name from dallas to the host name of the remote system on which the command is to be executed.
USER_ID	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 **joe**. This will need to be change to a valid user ID on the remote system identified by the REMOTE_HOST option.
USER_PASSWORD	Password for the user ID on the remote system. The examples use an arbitrary value of **abcdefg**. This will need to be changed to the password for the USER_ID you use to execute the remote command.

Executing Universal Command Manager on z/OS

Universal Command Manager is run as a batch job step on z/OS.

A UCMD Manager JCL procedure is provided with the Workload Automation installation to simplify JCL requirements. The JCL procedure name is **UCMDPRC**; it is located in the **SUNVSAMP** product library. See the Universal Command 5.1.0 Reference Guide for more information on the UCMDPRC procedure.

The following figure illustrates the JCL to execute UCMD Manager in a step. The input options are specified on the SYSIN ddname.

```
//S1 EXEC UCMDPRC
//SYSIN DD *
-cmd 'dir \' -host dallas -userid joe –pwd abcdefg
/*
```

You will need to make the following changes to this example so that it will run in your environment:

- Change the host name **dallas** specified with the -host option to the host name of the remote system on which to execute the command.
- Change the user ID **joe** to a valid user ID on the remote system.
- Change the password value **abcdefg** to the password for the user ID.

When UCMD Manager is executed, it will establish network connections with UCMD Server on the remote system named dallas, and provide the specified options to the UCMD Server. UCMD Server will execute the specified command ’dir \’ as user identifier **joe**.

The standard output of the remote command is written to the UCMD Manager UNVOUT ddname allocated in the **UCMDPRC** procedure. The standard error of the remote command is written to the UCMD Manager UNVERR ddname allocated in the **UCMDPRC** procedure. The default allocation for both UNVOUT and UNVERR is to SYSOUT. Similarly, standard input is allocated to the UNVIN ddname in the **UCMDPRC**. UNVIN is not utilized by the remote command being executed in this example.

The UCMD Manager will execute until the remote command completes and the UCMD Server sends the exit conditions of the remote command back to the UCMD Manager. The UCMD Manager will then end with the same exit code as the remote command.

Components

Universal Command Manager for z/OS

Executing Universal Command Manager on Windows

Universal Command Manager is run as a command on Windows.

The following command and command line options execute UCMD Manager.

```
ucmd -cmd 'dir \' -host dallas –userid joe –pwd password
```

You will need to make the following changes to this example so that it will run in your environment:

- Change the host name **dallas** specified with the -host option to the host name of the remote system on which to execute the command.
- Change the user ID **joe** to a valid user ID on the remote system.
- Change the password value **abcdefg** to the password for the user ID.

When UCMD Manager is executed, it will establish network connections with UCMD Server on the remote system named **dallas**, and provide the specified options to the UCMD Server. UCMD Server will execute the specified command ’dir \’ as user identifier **joe**.

The standard output of the remote command is written to the standard output of UCMD Manager, which is allocated to the console window. The standard error of the remote command is written to the standard error of the UCMD Manager, which is allocated to the console window. Similarly, standard input of the remote command is read from the standard input of the UCMD Manager, which is allocated to the console windows. Standard input is not utilized by the remote command being executed in this example.

The UCMD Manager will execute until the remote command completes and the UCMD Server sends the exit conditions of the remote command back to the UCMD Manager. The UCMD Manager will then end with the same exit code as the remote command.

Components

Universal Command Manager for Windows

Executing Universal Command Manager on UNIX

Universal Command Manager is run as a shell command on UNIX.

The following command and command line options execute UCMD Manager.

```
ucmd -cmd 'dir \' -host dallas –userid joe –pwd password
```

You will need to make the following changes to this example so that it will run in your environment:

- Change the host name **dallas** specified with the -host option to the host name of the remote system on which to execute the command.
- Change the user ID **joe** to a valid user ID on the remote system.
- Change the password value **abcdefg** to the password for the user ID.

The ucmd program is installed by default in directory **/opt/universal/bin**. This directory should be added to your PATH environment variable so that the shell can find the ucmd program. Alternatively, you can specify the full path name, **/opt/universal/bin/ucmd**.

When UCMD Manager is executed, it will establish network connections with UCMD Server on the remote system named dallas, and provide the specified options to the UCMD Server. UCMD Server will execute the specified command ’dir \’ as user identifier **joe**.

The standard output of the remote command is written to the standard output of UCMD Manager, which is allocated to the terminal. The standard error of the remote command is written to the standard error of the UCMD Manager, which is allocated to the terminal. Similarly, standard input of the remote command is read from the standard input of the UCMD Manager, which is allocated to the terminal. Standard input is not utilized by the remote command being executed in this example.

The UCMD Manager will execute until the remote command completes and the UCMD Server sends the exit conditions of the remote command back to the UCMD Manager. The UCMD Manager will then end with the same exit code as the remote command.

Components

Universal Command Manager for UNIX

Executing Universal Command Manager on IBM i

Universal Command Manager is run as a CL command on IBM i.

The following CL command and parameters execute UCMD Manager.

```
STRUCM CMD(‘dir \’) HOST(dallas) USERID(joe) PWD(abcdefg)
```

You will need to make the following changes to this example so that it will run in your environment:

- Change the host name **dallas** specified with the HOST option to the host name of the remote system on which to execute the command.
- Change the user ID **joe** to a valid user ID on the remote system.
- Change the password value **abcdefg** to the password for the user ID.

When UCMD Manager is executed, it will establish network connections with UCMD Server on the remote system named dallas, and provide the specified options to the UCMD Server. UCMD Server will execute the specified command ’dir \’ as user identifier **joe**.

The standard output and standard error of the remote command are written to the standard output and standard error, respectively, of UCMD Manager, which is allocated to the user's terminal for interactive sessions and to the printer file QPRINT for non-interactive jobs. Similarly, standard input of the remote command is read from the standard input of the UCMD Manager, which is allocated to the user's terminal for interactive sessions and to the QINLINE file for non-interactive jobs. Standard input is not utilized by the remote command being executed in this example.

The UCMD Manager will execute until the remote command completes and the UCMD Server sends the exit conditions of the remote command back to the UCMD Manager. The UCMD Manager will then end with an escape message if the exit condition was other than success.

Components

Universal Command Manager for IBM i

Executing Universal Command Manager on HP NonStop

Universal Command Manager is run as a TACL command.

The following command and command line options execute UCMD Manager.

```
run $SYSTEM.UNVBIN.ucmd –cmd 'dir \' -host dallas –userid joe –pwd password
```

You will need to make the following changes to this example so that it will run in your environment:

- Change the host name **dallas** specified with the -host option to the host name of the remote system on which to execute the command.
- Change the user ID **joe** to a valid user ID on the remote system.
- Change the password value **abcdefg** to the password for the user ID.

When UCMD Manager is executed, it will establish network connections with UCMD Server on the remote system named dallas, and provide the specified options to the UCMD Server. UCMD Server will execute the specified command ’dir \’ as user identifier **joe**.

The standard output of the remote command is written to the standard output of UCMD Manager, which is allocated to the terminal. The standard error of the remote command is written to the standard error of the UCMD Manager, which is allocated to the terminal. Similarly, standard input of the remote command is read from the standard input of the UCMD Manager, which is allocated to the terminal. Standard input is not utilized by the remote command being executed in this example.

The UCMD Manager will execute until the remote command completes and the UCMD Server sends the exit conditions of the remote command back to the UCMD Manager. The UCMD Manager will then end with the same exit code as the remote command.

Components

Universal Command Manager for HP NonStop

Remote Execution - Examples

Remote Execution Examples - z/OS

- Back up UNIX Directory to z/OS Dataset
- Restore UNIX Directory Backup from z/OS Dataset to UNIX Directory
- Directory Listing for UNIX Server from z/OS
- Directory Listing for Windows Server from z/OS
- Provide Network Status of Remote UNIX from z/OS
- Use UNIX tee Command to Store stdout to Local Server and z/OS
- Use an Encrypted Command File for User ID and Password on z/OS
- Override Standard z/OS IO File ddnames
- Override z/OS Standard Files with Procedure Symbolic Parameters
- Specifying UCMD for z/OS Options with the EXEC PARM
- Executing an Existing Windows .bat File from z/OS
- Using Manager Fault Tolerance from z/OS
- Restarting a Manager Fault Tolerant UCMD Manager on z/OS
- Automatically Create a Unique z/OS Command ID Using CA-Driver Variables
- Automatically Create a Unique z/OS Command ID Using Zeke Variables
- Automatically Create a Unique z/OS Command ID Using OPC Variables
- Universal Submit Job from z/OS to IBM i Using Remote Reply Facility
- Executing Universal Return Code within a Script via UCMD Manager for z/OS
- Executing URC and UMET within a Script via UCMD Manager for z/OS
- Using Encrypted Command File on z/OS

Remote Execution Examples - Windows

- Back up UNIX Directory to Windows
- Restore UNIX Directory Backup from Windows to UNIX
- Provide Network Status of Remote UNIX from Windows
- Redirect Standard Out and Standard Error to Windows
- Start UNIX Background Process from Windows
- Redirect Standard Input from Initiating System on Windows
- Universal Submit Job from Windows to IBM i
- Using Encrypted Command File on Windows

Remote Execution Examples - UNIX

- Provide Network Status of Remote Windows from UNIX
- Redirect Standard Out and Standard Error to UNIX
- Redirect Standard Input from Initiating System to UNIX
- Redirect Standard Input in UNIX Background Process
- Issue Universal Submit Job from UNIX to IBM i
- Using Encrypted Command File on UNIX

Remote Execution Examples - IBM i

- Provide Network Status of Remote Windows from IBM i
- Execute Script to Provide Network Status of Remote Windows from IBM i
- Display Library with Manager Fault Tolerance Active Using USBMJOB
- Universal Submit Job from zOS to IBM i
- Using Encrypted Command File on IBM i

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

Remote Execution Examples - HP NonStop

- Provide Network Status of Remote Windows from HP NonStop
- Execute Script to Provide Network Status of Remote Windows from HP NonStop

Back up UNIX Directory to zOS Dataset

Back up UNIX Directory to z/OS Dataset

This example demonstrates using UCMD Manager on z/OS to back up a UNIX directory, using the UNIX **tar** and **compress** commands, to a z/OS data set.

The backup script is allocated to the ddname **MYSCRIPT**. The script writes the **tar.Z** file to its standard out which is transmitted by the UCMD Server to the UCMD Manager which writes it to the UNVOUT ddname. The data set *hlq***.BKUP.TAR.Z** allocated to the UNVOUT ddname will contain the **tar.Z** backup file.

The data set *hlq***.BKUP.TAR.Z** must be a variable record format data set. Any valid record length or valid block size will work.

```
//jobname  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*
//         JCLLIB ORDER=SBI.UNV.SUNVSAMP
//*
//STEP1    EXEC UCMDPRC,
//         STDOUT='DISP=SHR,DSN=hlq.BKUP.TAR.Z'
//*
//MYSCRIPT DD  *
cd /export/home/username/fnd || exit 8
tar -cvzf - . || exit 8
//SYSIN    DD  *
-host   hostname
-userid  username
-pwd     password
-script  MYSCRIPT
-stdout  -mode binary
/*
```

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-host	Host name or IP address of the remote system on which to execute the script.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-script	ddname from which to read the script file. The script file is sent to the remote system by the UCMD Manager for execution.
-stdout	Starts the stdout options. All options read afterwards are applied to the stdout file. The first option not recognized as a standard file option terminates the stdout option list.
-mode	Transfer mode for the stdout file: **binary**. Since a backup file is a tar, compressed format contains binary data, it should not be translated as text data.

Components

Universal Command Manager for z/OS

Restore UNIX Directory Backup from zOS Dataset to UNIX Directory

Restore UNIX Directory Backup from z/OS Dataset to UNIX Directory

This example demonstrates using UCMD Manager on z/OS to restore a directory on a UNIX system from a tar.Z backup maintained on the z/OS system. See Back up UNIX Directory to zOS Dataset to see how the backup data set was created.

The UNIX script uses the **tar** command to extract the files to be restored from the **tar.Z** backup. The **tar** command is directed to read the **tar.Z** file from its standard input with the tar command line option **-f –**, which results in it reading from the UCMD Manager UNVIN ddname. The Manager UNVIN ddname allocates the **tar.Z** backup data set that was created previously.

```
//jobname  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*
//  JCLLIB ORDER=SBI.UNV.SUNVSAMP
//*
//STEP1  EXEC UCMDPRC,
//  STDIN='DISP=SHR,DSN=hlq.BKUP.TAR.Z'
//MYSCRIPT DD *
# Check if the directory exists.  If it does not, create it.
  if test ! -d /export/home/username/fnd
     then mkdir /export/home/username/fnd || exit 8
  fi
  cd /export/home/username/fnd || exit 8
# Note: Not all tar commands recognize the 'B' argument.  If you
# receive an error message indicating this from the remote
# UNIX system, remove the 'B' argument.
# The 'B' argument is used to force tar to read multiple
# times to fill the block.

tar -xzvBf - || exit 8

//SYSIN DD *
-script myscript
-host   hostname
-userid username
-pwd    password
-stdin -mode binary
/*
```

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	ddname from which to read the script file. The script file is sent to the remote system by UCMD Manager for execution.
-host	Host name or IP address of the remote system on which to execute the script
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

-stdin	Starts the stdin options. All options read afterwards are applied to the stdin file. The first option not recognized as a standard file option terminates the stdin option list.
-mode	Transfer mode for the stdin file: **binary**. Since a backup file is a tar, compressed format contains binary data, it should not be translated as text data.

Components

Universal Command Manager for z/OS

Directory Listing for UNIX Server from zOS

Directory Listing for UNIX Server from z/OS

This example demonstrates executing a simple UNIX command from z/OS using the COMMAND option, as opposed to the SCRIPT option.

The UNIX **ls** command is executed in the example. The **ls** command writes the file listing to its standard out, which is redirected to the UCMD Manager on z/OS. The UCMD Manager writes the standard out to ddname UNVOUT.

```
//jobname  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*
//       JCLLIB ORDER=SBI.UNV.SUNVSAMP
//*
//STEP1    EXEC UCMDPRC
//SYSIN    DD *
-cmd "ls -l /opt/universal/bin"
-host hostname
-userid username
-pwd password
/*
```

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-cmd	Command to be executed on the remote system.
-host	Host name or IP address of the remote system on which to execute the script.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

Components

Universal Command Manager for z/OS

Directory Listing for Windows Server from zOS

Directory Listing for Windows Server from z/OS

This example demonstrates executing a simple Windows command from z/OS using the COMMAND option, as opposed to the SCRIPT option.

The Windows DIR command is executed in this example. The DIR command writes the file listing to its standard out, which is redirected to the UCMD Manager on z/OS. The UCMD Manager writes the standard out to ddname UNVOUT.

```
//jobname  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*
//         JCLLIB ORDER=SBI.UNV.SUNVSAMP
//*
//STEP1    EXEC UCMDPRC
//SYSIN    DD *
-cmd    ’dir \’
-host   hostname
-userid userid
-pwd    password
/*
```

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-cmd	Command to be executed on the remote system.
-host	Host name or IP address of the remote system on which to execute the script.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

Components

Universal Command Manager for z/OS

Provide Network Status of Remote UNIX from zOS

Provide Network Status of Remote UNIX from z/OS

This example demonstrates executing a simple UNIX command from z/OS using the SCRIPT option, as opposed to the COMMAND option.

The UNIX **netstat** command is executed in the example. The **netstat** command writes the network status report to its standard out, which is redirected to the UCMD Manager on z/OS. The UCMD Manager writes the standard out to ddname UNVOUT.

```
//jobname JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*
//        JCLLIB ORDER=SBI.UNV.SUNVSAMP
//*
//STEP1   EXEC UCMDPRC
//SCRIPT  DD *
netstat
//SYSIN   DD *
-host     hostname
-userid   username
-pwd      password
-script   SCRIPT
/*
```

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-host	Host name or IP address of the remote system on which to execute the script.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-script	ddname from which to read a script file. The script file is sent to the remote system by UCMD Manager for execution.

Components

Universal Command Manager for z/OS

Use UNIX tee Command to Store stdout to Local Server and zOS

Use UNIX tee Command to Store stdout to Local Server and z/OS

This example demonstrates using the UNIX **tee** command to save the standard out of a command to a file on the remote system, as well as send it back to the UCMD Manager in real time.

The UNIX **ls** command is executed in this example. The **ls** command writes the file listing to its standard out, which is piped to the **tee** command. The **tee** command reads the **ls** listing and writes it to both the file **teeout.txt** and to standard out, which is redirected to the UCMD Manager on z/OS. The UCMD Manager writes the standard out to ddname UNVOUT.

```
//jobname  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*
//         JCLLIB ORDER=SBI.UNV.SUNVSAMP
//*
//STEP1 EXEC UCMDPRC
//SCRIPT DD *
  ls | tee teeout.txt
//SYSIN DD *
-script   SCRIPT
-host     hostname
-userid   username
-pwd      password
/*
```

SYSIN Options

Option	Description
-script	ddname from which to read a script file. The script file is sent to the remote system by UCMD Manager for execution
-host	Host name or IP address of the remote system on which to execute the script.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

Components

Universal Command Manager for z/OS

Use an Encrypted Command File for User ID and Password on zOS

Use an Encrypted Command File for User ID and Password on z/OS

This example demonstrates using an encrypted command file to hold the user ID and password options used by UCMD Manager for z/OS. The encrypted command file is created with the Opswise Automation Center Universal Encrypt (UENCRYPT) utility.

Create an Encrypted Command File

The following figure demonstrates how to create a UENCRYPTed command file. The options to be encrypted are specified on the UNVIN ddname (in this example, **-userid** and **-pwd**). The encrypted command file is written to ddname UNVOUT, which allocates PDS member **USR001**.

```
//jobname   JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*
//STEP1   EXEC UENCRYPT
//STEPLIB  DD DISP=SHR,DSN=SBI.UNV.SUNVLOAD
//UNVIN    DD *
-userid username
-pwd    password
//UNVOUT   DD DISP=SHR,DSN=hlq.PROD.DATA(USR001)
//UNVNLS   DD DISP=SHR,DSN=SBI.UNV.SUNVNLS
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//CEEDUMP  DD SYSOUT=*
//SYSIN    DD DUMMY
```

UNVIN Options

The UNVIN 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.

Use an Encrypted Command File

The following figure demonstrates how to use an encrypted command file to execute a remote command using UCMD Manager. The example is executing a UNIX **ls** command on a remote system. The user ID and password to be used to execute the command is specified with the **-encryptedfile** option. The **-encryptedfile** option specifies a ddname from which to read the encrypted command file created in the first figure.

```
//jobname  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*
//         JCLLIB ORDER=SBI.UNV.SUNVSAMP
//*
//STEP1  EXEC UCMDPRC
//USER    DD DISP=SHR,DSN=hlq.PROD.DATA(USR001)
//SYSIN   DD *
-cmd   'ls -la'
-host  hostname
-encryptedfile USER
/*
```

⚠ **Note**
An encrypted command file protects the privacy of the data contained within it. It does not protect it from being used by anyone with read permission to the encrypted command file data set.

To ensure the encrypted command file is used only by authorized users, proper security access to the data set must be defined in your security system, such as IBM's RACF.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-cmd	Command to be executed on the remote system.
-host	Host name or IP address of the remote system on which to execute the script.
-encryptedfile	ddname from which to read an encrypted command file created with the Opswise Automation Center Universal Encrypt (UENCRYPT) utility.

Components

Universal Command Manager for z/OS

Override Standard zOS IO File ddnames

Override Standard z/OS I/O File ddnames

This example demonstrates how to override the z/OS standard output and standard error ddnames in the UCMDPRC procedure.

The example:

- Overrides the UNVOUT ddname in the UCMDPRC procedure with a data set allocation.
- Overrides the UNVERR ddname with a SYSOUT class of H.

```
//jobname JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*
// JCLLIB ORDER=SBI.UNV.SUNVSAMP
//*
//STEP1  EXEC UCMDPRC
//UNVOUT  DD  DISP=SHR,DSN=hlq.APP.LIST(OUTPUT)
//* UNIVERSAL COMMAND WILL CREATE THE MEMBER
//UNVERR  DD  SYSOUT=H
//SYSIN   DD  *
-host   hostname
-userid username
-pwd    password
-cmd    command
/*
```

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-host	Host name or IP address of the remote system on which to execute the script.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-cmd	Command to be executed on the remote system.

Components

Universal Command Manager for z/OS

Override zOS Standard Files with Procedure Symbolic Parameters

Override z/OS Standard Files with Procedure Symbolic Parameters

This example demonstrates how to override the z/OS standard input, output, and error using the UCMDPRC procedure symbolic parameters.

The UCMDPRC procedure provides parameters STDIN, STDOUT, and STDERR for the allocation of the UNVIN, UNVOUT, and UNVERR ddnames, respectively.

```
//jobname JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*
//        JCLLIB ORDER=SBI.UNV.SUNVSAMP
//*
//STEP1 EXEC UCMDPRC,
//  STDIN='DUMMY',
//  STDERR='SYSOUT=H',
//  STDOUT='DISP=SHR,DSN=hlq.DATA.LIST(OUT2)'
//SYSIN  DD  *
-host     hostname
-userid   username
-pwd      password
-cmd      command
/*
```

SYSIN Options

Option	Description
-host	Host name or IP address of the remote system on which to execute the script.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-cmd	Command to execute on the remote system.

Components

Universal Command Manager for z/OS

Specifying UCMD for zOS Options with the EXEC PARM

Specifying UCMD for z/OS Options with the EXEC PARM

This example demonstrates how to specify UCMD options using the EXEC statement PARM keyword.

UCMD Manager reads its options typically from the SYSIN ddname, but options can be specified on the EXEC statement PARM keyword as well. Options specified as PARM values override options specified on the SYSIN ddname.

The UCMDPRC JCL procedure provides the symbolic parameter UPARM to specify the options. The example sets the UCMD Manager message level to a value of **audit** using the **-level** option.

```
//jobname JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*
//        JCLLIB ORDER=SBI.UNV.SUNVSAMP
//*
//STEP1 EXEC UCMDPRC,UPARM='-level audit'
//SYSIN DD *
-cmd command -host hostname -userid username -pwd password
```

SYSIN Options

Option	Description
-cmd	Command to execute on the remote system.
-host	Host name or IP address of the remote system on which to execute the script.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

Components

Universal Command Manager for z/OS

Executing an Existing Windows .bat File from zOS

Execute an Existing Windows .bat file from z/OS

This example demonstrates calling a Windows batch file (**.bat** extension) from a script being executed remotely by UCMD Manager. The Windows **CALL** command is used to execute a Windows batch file that already exists on the Windows system.

When UCMD Manager is provided a script with the **-script** option to be remotely executed, it sends the script to the remote UCMD Server. The UCMD Server, in turn, will create a temporary Windows batch file (**.bat** extension) in the file system to hold the UCMD Manager-provided script. The UCMD Server will then execute the saved batch file using the Windows command processor **CMD.EXE**.

Since the UCMD Manager script is executing as a Windows batch file, if it needs to call another batch file, it must use the Windows **CALL** command.

```
//jobname JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//        JCLLIB ORDER=SBI.UNV.SUNVSAMP
//*
//STEP1 EXEC UCMDPRC
//SCRIPTDD DD *
call user.bat
//SYSIN DD *
-script SCRIPTDD -host hostname -userid username -pwd password
/*
```

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	ddname from which to read the script file. The script file is sent to the remote system by UCMD Manager for execution.
-host	Host name or IP address of the remote system on which to execute the script.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

Components

Universal Command Manager for z/OS

Using Manager Fault Tolerance from zOS

Using Manager Fault Tolerance from z/OS

This example demonstrates using the Manager Fault Tolerant (MFT) feature of UCMD.

The UCMD Manager will execute until the work being executed remotely ends. If the UCMD Manager would end prematurely (for example, if it was canceled), the UCMD Server would also end and, in the process, terminate the work it was executing on behalf of the Manager. If MFT is used, the UCMD Manager can end and the UCMD Server will continue to execute until the work completes.

MFT requires that the UCMD Server is configured to allow for spooling of standard I/O files. The UCMD Server option to use for this configuration is ALLOW_SPOOLING. Its default value is **no**; it must be set to **yes**.

A UCMD Manager specifies the use of MFT by setting the **-managerft** option (MANAGER_FAULT_TOLERANT) to a value of **yes**. Additionally, MFT requires a command identifier that uniquely identifies the work to be executed on the remote system. The command ID is a text value of your choosing. An example command ID is **example-2010-07-02** or it could be automatically generated for you by UCMD Manager. In either case, if the Manager prematurely ends, the command ID will be required to restart the Manager and complete the work executed on the remote system.

```
//jobname  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//         JCLLIB ORDER=SBI.UNV.SUNVSAMP
//*
//STEP1 EXEC UCMDPRC
//SYSIN DD *
-cmd    command
-host   hostname
-userid username -pwd password
-managerft yes  -cmdid  example-2010-07-02
```

See Indesca - Fault Tolerance Implementation for complete details on the Manager Fault Tolerant feature.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-cmd	Command to execute on the remote system.
-host	Host name or IP address of the remote system on which to execute the script.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-managerft	Specification for whether or not the Manager Fault Tolerance (MFT) feature is used. A value of **yes** specifies that MFT should be used.
-cmdid	Unique command ID associated with the remote unit of work.

Components

Universal Command Manager for z/OS

Restarting a Manager Fault Tolerant UCMD Manager on zOS

Restarting a Manager Fault Tolerant UCMD Manager on z/OS

In Using Manager Fault Tolerance from zOS, the example demonstrates how to remotely execute work using the Manager Fault Tolerant (MFT) feature.

The following example demonstrates how to restart a UCMD Manager that premature ended when using MFT.

If a UCMD Manager executing with MFT ends prematurely, the UCMD Server and the remote work will continue executing until the remote work has completed. All standard I/O files are saved on the UCMD Server system, along with the exit conditions of the work. They will remain on the UCMD Server system until a UCMD Manager is restarted using the same command ID that identifies the work.

A restart can be performed after the remote work has complete or while the remote work is still in executing.

Continuing from Using Manager Fault Tolerance from zOS, the following example illustrates a UCMD Manager restart for the work identified by command ID **example-2010-07-02**.

```
//jobname  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*
//         JCLLIB ORDER=SBI.UNV.SUNVSAMP
//*
//STEP1 EXEC UCMDPRC
//SYSIN DD *
-cmd command -host hostname -userid username -pwd password
-managerft yes -cmdid example-2010-07-02
-restart yes
/*
```

SYSIN Options

Option	Description
-cmd	Command to be executed on the remote system.
-host	Host name or IP address of the remote system on which to execute the command.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-managerft	Specification for whether or not the manager fault tolerant feature is used.
-cmdid	Unique command ID associated with the remote unit of work.
-restart	Specification for whether or not the manager is requesting restart. A value of **yes** indicates the Manager is being restarted.

Components

Universal Command Manager for z/OS

Automatically Create a Unique zOS Command ID Using CA-Driver Variables

Automatically Create a Unique z/OS Command ID Using CA-Driver Variables

CA provides variable functionality via CA-Driver. (Refer to the CA manuals for instructions on variable usage.)

CA-Driver variables specific to CA-7, such as the following, can be used to create a unique **cmdid**. (Refer to the CA documentation for a complete list of CA-Driver variables available for CA-7 and CA-Scheduler.)

- &C_L2JN is the CA-7 Job Name
- &C_L27# is the CA-7 Job Number
- &C_L2DOD is the Due Out Date for this Job execution.

One Procedure should be placed in the CA-Driver Procedure library.

The following example illustrates a CA-Driver Procedure using the above system variables to create a unique command ID.

```
//DRVRUCMD    DPROC REMOTEJOBNAME=UCMD
-cmdid ‘&REMOTEJOBNAME.&C_L2JN.&C_L27#.&C_L2DOD’
```

Each step that executes UCMD should reference this procedure in order to create the unique UCMD **cmdid** as the first parameter within the UCMD SYSIN DD statement. This procedure defaults the **cmdid** to the values defined by one user variable called **remotejobname** and 3 CA-7 variables.

(See the following examples.)

Example 1

The following example illustrates how to call the PROC **DRVRUCMD** from within the UCMD SYSIN DD statement.

The variables set in the **DRVRUCMD** PROC are set to the following for this example:

- &C_L2JN = PRD00001
- &C_L27# = 0030001
- &C_L2DOD 03265
- &REMOTEJOBNAME = UCMD (default value in Driver Procedure **DRVRUCMD**)

```
//S1 EXEC UCMDPRC
//SYSIN DD *
//CALL EXEC  PROC=DRVRUCMD,REMOTEJOBNAME=
-cmd command -host hostname -userid username -pwd password
-managerft yes -restart auto
/*

Expanded Results:

//S1 EXEC UCMDPRC
//SYSIN DD *
-cmdid UCMDPRD000010030001032625
-cmd command -host hostname -userid username -pwd password
-managerft yes -restart auto
/*
```

Note
If the **cmdid** identifier contains spaces, it must be enclosed in either single (**'**) or double (**"**) quotation marks.

Example 2

The following example illustrates how to override the variable value for **REMOTEJOBNAME** in the **CALL** step.

```
//S1    EXEC UCMDPRC
//SYSIN DD *
//CALL  EXEC  PROC=DRVRUCMD,REMOTEJOBNAME=unixpayrolljob1
-cmd command -host hostname -userid username -pwd password
-managerft yes -restart auto
/*

Expanded Results:

//S1    EXEC UCMDPRC
//SYSIN DD *
-cmdid unixpayrolljob1PRD000010030001032625
-cmd command -host hostname -userid username -pwd password
-managerft yes -restart auto
/*
```

Note
If the **cmdid** identifier contains spaces, it must be enclosed in either single (**'**) or double (**"**) quotation marks.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-cmd	Command to be executed on the remote system.
-host	Host name or IP address of the remote system on which to execute the command.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-managerft	Specification for whether or not the manager fault tolerant feature is used.
-cmdid	Unique command ID associated with the remote unit of work.
-restart	Specification for whether or not the manager is requesting restart. A value of **yes** indicates the Manager is being restarted.

Components

Universal Command Manager for z/OS

Automatically Create a Unique zOS Command ID Using Zeke Variables

Automatically Create a Unique z/OS Command ID for your Indesca Process Using Zeke Variables

Zeke has a set of reserved variables available that get substituted during job submission. The default character **$** is used to identify a Zeke variable within the instream JCL. This default character can be changed during installation. Create a variable whose value is set based on the current schedule date. Use this variable in the UCMD Manager jobs.

(See the ASG Zeke documentation for instructions on variable usage.)

```
//S1 EXEC UCMDPRC
//LOGONDD  DD DISP=SHR,DSN=hlq.encrypted.file
//SCRIPTDD DD *
DIR
//SYSIN      DD *
-cmdid ‘jobname$SCHDATEunixpayrolljob1’                                              <== Zeke
variable
-script scriptdd –host dallas –encryptedfile logondd –managerft yes –restart auto
/*

Expanded Results:

//S1 EXEC UCMDPRC
//LOGONDD  DD DISP=SHR,DSN=hlq.encrypted.file
//SCRIPTDD DD *
dir
//SYSIN      DD *
-cmdid ‘JOBNAME03254unixpayrolljob1’
-script scriptdd –host dallas –encryptedfile logondd –managerft yes –restart auto
/*
```

* If the **cmdid** identifier contains spaces, it must be enclosed in either single (**’**) or double (**"**) quotation marks.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-cmdid	Unique command ID associated with the remote unit of work.
-script	ddname from which to read a script file. The script file is sent to the remote system by UCMD Manager for execution.
-host	Host name or IP address of the remote system on which to execute the command.
-encryptedfile	ddname from which to read an encrypted command file created with the Opswise Automation Center Universal Encrypt (UENCRYPT) utility.
-managerft	Specification for whether or not the manager fault tolerant feature is used.

-restart	Specification for whether or not the manager is requesting restart. A value of **yes** indicates the Manager is being restarted.

Components

Universal Command Manager for z/OS

Automatically Create a Unique zOS Command ID Using OPC Variables

Automatically Create a Unique z/OS Command ID for your Indesca Process Using OPC Variables

OPC has a set of reserved variables available that get substituted at job submission.

The feature gets switched on by coding the following JCL statements:

- //*%OPC SCAN   <== set substitution on and off by
- //*%OPC NOSCAN   <== set substitution off

Any OPC variable found within the instream JCL can be substituted with the current value by OPC. (See the IBM OPC documentation for instructions on variable usage.)

```
//S1 EXEC UCMDPRC
//LOGONDD  DD DISP=SHR,DSN=hlq.encrypted.file
//SCRIPTDD DD *
DIR
SYSIN    DD *
-cmdid payrolljob&CYMD.&CHHMMSSX.                                                     <== OPC
variables
-script scriptdd –host dallas –encryptedfile logondd –managerft yes –restart no
/*

Expanded Results:

//S1 EXEC UCMDPRC
//LOGONDD  DD DISP=SHR,DSN=hlq.encrypted.file
//SCRIPTDD DD *
dir
//SYSIN    DD *
-cmdid payrolljob2003061613315614
-script scriptdd -host dallas –encryptedfile logondd –managerft yes –restart no
/*
```

* If the **cmdid** identifier contains spaces, it must be enclosed in either single (**'**) or double (**"**) quotation marks.

SYSIN Options

The SYSIN options used in this example are:

Option	**Description**
-cmdid	Unique command ID associated with the remote unit of work.
-script	ddname from which to read a script file. The script file is sent to the remote system by UCMD Manager for execution.
-host	Host name or IP address of the remote system on which to execute the command.
-encryptedfile	ddname from which to read an encrypted command file created with the Opswise Automation Center Universal Encrypt (UENCRYPT) utility.

-managerft	Specification for whether or not the manager fault tolerant feature is used.
-restart	Specification for whether or not the manager is requesting restart. A value of **yes** indicates the Manager is being restarted.

Components

Universal Command Manager for z/OS

Universal Submit Job from zOS to IBM i Using Remote Reply Facility

Universal Submit Job from z/OS to IBM i using the Remote Reply Facility

This example demonstrates how to submit an IBM i batch job from z/OS and use the Remote Reply Facility.

Native IBM i SBMJOB parameters can be specified as part of the Universal Submit Job **USBMJOB** command. The Remote Reply Facility detects messages, issued by the submitted job, that require a reply. The message then will be passed to a remote z/OS system for a reply. When the reply is received, the reply will be sent to the IBM i message queue that is waiting for the reply.

z/OS issues the message to the z/OS console as a WTOR (Write To Operator with Reply) message. The WTOR message is written to the z/OS console using the Stonebranch USS **uwto** command. The reply to the message is sent back to the IBM i system.

```
//S1    EXEC UCMDPRC
//SCRIPT   DD *
ADDLIBLE LIB(UNVPRD510)
UNVPRD510/USBMJOB CMD(dsplib ibmi-username) +
RMTRPY(*YES) +
RMTREFRESH(60) +
RMTMSGPRFX('TESTPRFX') +
RMTHOST(zos-hostname) +
MSGCMDPATH("usrlocaluniversalbinuwto") +
RMTUSER(zos-username) + RMTPWD(zos-password)
//SYSIN DD *
-script SCRIPT -host ibmi-hostname -userid ibmi-username -pwd ibmi-password
/*
```

This UCMD Manager executes the script on host called **ibmi-hostname**. The IBM i user ID **ibmi-username** and password **ibmi-password** are used for authentication on the IBM i system. The script runs with the authority of user **ibmi-username**.

The reply message, should there be one, is sent to the host name **zos-hostname** for a reply. The z/OS USS **uwto** command runs with user ID **zos-username** and password **zos-password**.

The first line of the script will add the library **UNVPRD510** to the library concatenation. The second line will execute the command **dsplib ibmi-username** with the USBMJOB utility. All output created by the command will be spooled to stdout of the manager job.

The Remote Reply Facility is turned on with the **RMTRPY** parameter; therefore, USBMJOB will send all messages requiring a reply to the remote z/OS console on host **zos-hostname**, as specified on the **RMTHOST** parameter. Replies to the inquiry messages are received from the z/OS console and sent to the IBM i message queue waiting for the reply.

The z/OS USS UWTO command is executed with the authority of the z/OS user **zos-userid** and **zos-password**, as specified by the **RMTUSER** and **RMTPWD** parameters, respectively. The z/OS console message is prefixed with **TESTPRFX**, as specified by the **RMTMSGPRFX** parameter.

If a response is not received within 60 seconds, the WTOR will be deleted and a new one sent, as specified by the **RMTREFRESH** parameter. The UWTO executable is found on the z/OS USS system at **/usr/local/universal/bin/uwto**, as specified by the **MSGCMDPATH** parameter. If **uwto** is in the USS system PATH, **MSGCMDPATH** is not required.

USBMJOB Options

The USBMJOB options used in this example are:

Option	Description
CMD	Specifies a command that runs in the submitted batch job. The command can be a maximum of 3000 characters.

RMTRPY	Specifies whether USBMJOB will use the Remote Reply Facility.
RMTREFRESH	Specifies the number of seconds to refresh the z/OS console message if no reply is received. The previous message is deleted from the console and a new one is issued.
RMTMSGPRFX	Specifies a text string up to 12 characters that will prefix the message written to the z/OS console.
RMTHOST	Specifies the host name or IP address of the z/OS system on which the reply message is issued.
MSGCMDPATH	Specifies the path name of the z/OS USS **uwto** program. If the **uwto** program is in the system PATH, this parameter is not required.
RMTUSER	Specifies the z/OS user ID with which the **uwto** program is executed.
RMTPWD	Specifies the password for the z/OS user ID specified with the RMTUSR parameter.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	ddname from which to read a script file. The script file is sent to the remote system by UCMD Manager for execution.
-host	Host name or IP address of the remote system on which to execute the script.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the remote user ID.

Components

Universal Command Manager for z/OS

Universal Submit Job

Executing Universal Return Code within a Script via UCMD Manager for zOS

Executing Universal Return Code within a Script via Universal Command Manager for z/OS

Note
As of Windows 2000, the Universal Return Code command is no longer necessary in Windows batch files. Microsoft added the ability to specify a script return code as an argument to the EXIT command in Windows 2000 and above.

This example illustrates the use of Universal Return Code to exit the script with a specific return code value.

By default, the return code of the last command within the script sets the return code of the script. Universal Return Code is useful when multiple commands are executed within one script.

The following example executes a Windows batch file as a script. The script executes the backup.exe program and saves its return code value in the variable RC. URC is then used to set the **ERRORLEVEL** value back to the saved RC value before exiting. A user variable called RC is set to the value of the **ERRORLEVEL** of the previous command.

```
//S1 EXEC UCMDPRC
//SCRIPTDD DD *
backup.exe > c:\temp\bkup.log
SET RC=%ERRORLEVEL%
UCOPY c:\temp\bkup.log
DEL c:\temp\bkup.log
URC %RC%
/*
//SYSIN    DD *
-host dallas
-script SCRIPTDD
-userid joe
-pwd abcdefg
/*
```

The first command executes a backup script. The next line sets a variable called RC to the value of the return code of the **backup.exe**.

The **UCOPY** command copies the log file to the Universal Command Manager. The next step deletes the log file.

The last line of the script then uses the variable **RC** as the URC value in order to set the return code of the script equal to the exit code of the **backup.exe** execution, instead of the return code of the **DEL** command.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-host	Host name or IP address of the remote system on which to execute the script.
-script	ddname from which to read the script file. The script file is sent to the remote system by the UCMD Manager for execution.
-userid	Remote user ID with which to execute the command.

-pwd	Password for the user ID.

Components

Universal Command Manager for z/OS

Universal Return Code

Executing URC and UMET within a Script via UCMD Manager for zOS

Executing Universal Return Code and Universal Message Translator within a Script via Universal Command Manager for z/OS

Note
As of Windows 2000, the Universal Return Code command is no longer necessary in Windows batch files. Microsoft added the ability to specify a script return code as an argument to the EXIT command in Windows 2000 and above.

The following example builds onto the Executing Universal Return Code within a Script via UCMD Manager for zOS example by adding a step that executes the Universal Message Translator (UMET) utility.

UMET could be used if the first command does not set the return code properly. The example exits with the return code of a command in the middle of the script with the use of Universal Return Code. A user variable called RC is set to the value of the return code of the UMET execution. The last line of the script then uses that value as the URC value to set the return code of the script equal to the exit code of the UMET execution.

```
//S1 EXEC UCMDPRC
//SCRIPTDD DD *
backup.exe > c:\temp\bkup.log
umet -table c:\temp\translate.table -file c:\temp\bkup.log
SET RC=%ERRORLEVEL%
UCOPY c:\temp\bkup.log
DEL c:\temp\bkup.log
URC %RC%
/*
//SYSIN    DD *
-host dallas
-script SCRIPTDD
-userid joe
-pwd abcdefg
/*
```

The first command executes a backup script. The second command executes the UMET program and sets the return code of UMET based on the table definitions and the file being interrogated. The next line sets a variable called RC to the value of the return code of the UMET execution. The UCOPY command copies the log file to the Universal Command Manager. The next line deletes the log file. The last line of the script then uses the variable RC as the URC value in order to set the return code of the script equal to the return code of the UMET execution instead of the return code of the DEL command.

Script Options

The script options used in this example are:

Option	Description
-table	Translation table file name.
-file	Input message file name.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-host	Host name or IP address of the remote system on which to execute the script.
-script	ddname from which to read the script file. The script file is sent to the remote system by the UCMD Manager for execution.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

Components

Universal Command Manager for z/OS

Universal Return Code

Universal Message Translator

Back up UNIX Directory to Windows

Back up UNIX Directory to Windows

This example backs up a directory and its subdirectories on a UNIX system to a local file. Instead of executing a command on the remote host, a local script file is executed.

The following figure illustrates the script in a file named **myscript**.

```
cd /usr/man/man1
tar -cv . | compress
```

The following figure illustrates the command to execute the script in **myscript**.

```
ucmd –script myscript -host dallas -userid joe -pwd password –stdout –mode binary > data.tar
```

The script file changes its current directory to the directory to backup. The **tar** command creates an archive file containing all files and subdirectories located in the current directory. This archive file is written to **tar**'s standard out, which is piped to the **compress** command. The compress command compresses its input and writes to its standard out. The standard out of the compress command is the same standard out of the script file. The script file's standard out is redirected back to the **ucmd** command running on the local system. The standard out of UCMD is redirected to the local file **data.tar**.

Command Line Options

The command line options used in this example are:

Option	Description
-script	File name of a script file. The script file is sent to the remote system for execution.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the command.
-pwd	Specifies the password for the user ID.
-stdout	Starts the stdout option list. All options read afterwards are applied to the stdout file. The first option not recognized as a standard file option terminates the stdout option list.
-mode	Transfer mode for the stdout file: **binary**. The data is not translated.

Components

Universal Command Manager for Windows

Restore UNIX Directory Backup from Windows to UNIX

Restore UNIX Directory Backup from Windows to UNIX

This example restores a directory that was backed up (see Back up UNIX Directory to Windows). The file containing the backup is on the local system.

The script is located in local file **myscript**.

The following figure illustrates the script to perform the restore.

```
if test ! -d man1
   then
      mkdir man1
   fi

cd man1
uncompress | tar –xvf -
diff . usrmanman1
```

The following figure illustrates the command to execute the script file.

```
ucmd -script myscript –host dallas –userid joe -pwd password -stdin –mode binary < file.tar
```

The script file creates directory **man1** in **joe**'s home directory if it does not already exist. It then changes its current directory to **man1**. The **uncompress** command reads from the script's standard in file, which is redirected from UCMD's standard in on the local system.

Notice that UCMD's standard in is redirected from the backup file **file.tar**. The **uncompress** program uncompresses its input and writes it to its standard out, which is piped to the **tar** command. The **tar** command extracts and writes the archive to the current directory. The final command, **diff**, compares the original directory with the new one. The **diff** command returns **0** if no differences are found; otherwise, it returns **1**.

Command Line Options

The command line options used in this example are:

Option	Description
-script	File name of a script file. The script file is sent to the remote system for execution.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

-stdin	Starts the stdin option list. All options read afterwards are applied to the stdin file. The first option not recognized as a standard file option terminates the stdin option list.
-mode	Transfer mode for the stdout file: **binary**. The data is not translated.

Components

Universal Command Manager for Windows

Provide Network Status of Remote UNIX from Windows

Provide Network Status of Remote UNIX from Windows

This example produces a report of the system status of a remote UNIX system. Instead of executing a command on the remote host, a local script file is executed.

The following figure illustrates the script file **myscript**.

Note
The commands executed in the script file may or may not require modifications depending on the type of UNIX system on which it executes.

```
echo "System Status as of `date`"
echo "--------------------------------------------------------------------------------------------"
netstat
echo "--------------------------------------------------------------------------------------------"
df
echo "--------------------------------------------------------------------------------------------"
ps –ax
```

The following figure illustrates the command to execute the script file.

```
ucmd -script myscript -host dallas -userid joe –pwd password
```

The report is written to the standard out of UCMD.

Command Line Options

The command line options used in this example are:

Option	Description
-script	File name of a script file. The script file is sent to the remote system for execution.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

Components

Universal Command Manager for Windows

Redirect Standard Out and Standard Error to Windows

Redirect Standard Out and Standard Error to Windows

The following example illustrates how to redirect the standard output and error of the ‘DIR’ command to a file on the initiating system.

```
ucmd -cmd "dir" -host dallas –userid joe –pwd password > output.file 2>&1
```

The command **dir** is sent to a remote system named **dallas** for execution. The standard output and standard error of the **dir** command are written back to the UCMD process and redirected to standard out file **output.file**. The process will authenticate and run under the authority of userid **joe**.

If the remote system is a UNIX system, change the command **dir** to **ls**.

Command Line Options

The command line options used in this example are:

Option	Description
-cmd	Remote command to execute.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

Components

Universal Command Manager for Windows

Start UNIX Background Process from Windows

Start UNIX Background Process from Windows

A UCMD Manager job will not end until the remote process ends and all standard files are closed. If the remote process starts a child process, UCMD Manager also will wait until the child process ends and its standard I/O files are closed.

In order to start the process without waiting for the process to end and close its standard I/O files, start the process in the background using the **nohup** command and redirect standard out and error to **/dev/null**.

```
ucmd -cmd "nohup startprocess > /dev/null 2>&1 &'" –host dallas –userid joe –pwd password
```

The command to start a process is issued with the UNIX **nohup** parameter. Any output is written to **/dev/null** which never is saved to disk or memory. The process will authenticate and run under the authority of userid **joe**.

Command Line Options

The command line options used in this example are:

Option	Description
-cmd	Remote command to execute.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

Components

Universal Command Manager for Windows

Redirect Standard Input from Initiating System on Windows

Redirect Standard Input from Initiating System on Windows

The **ucmd** command reads from standard input and writes it to the UCMD Server for the remote command to read as its standard input. The allocation of standard input can be changed with a shell redirection operator. The redirection operators instruct the shell to change the allocation of the standard files. To change the allocation of standard input, use the **<** operator.

```
ucmd –script myscript -host dallas –userid joe –pwd password < input.file
```

The command is sent to a remote system named **dallas** for execution. The output of the script is redirected back to the Universal Command process's standard out and standard error. Standard input is read from file **input.file** on the initiating system. The process will authenticate and run under the authority of userid **joe**.

Command Line Options

The command line options used in this example are:

Option	Description
-script	File from which to read a script file. The script file is sent to the remote system for execution.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

Components

Universal Command Manager for Windows

Universal Submit Job from Windows to IBM i

Universal Submit Job from Windows to IBM i

The following example illustrates the issuing of a command from Windows to remote IBM i as a parameter of USBMJOB.

```
ucmd -cmd "usbmjob cmd(dspsyssts)" -host ohio -userid usrid -pwd usrpwd
```

In this example, USBMJOB is submitted to the server running on the host **ohio**.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-cmd	Universal Command command option (**usbmjob**).
-host	Directs the command to a computer with a host name of **ohio**.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the remote user ID.

Components

Universal Command Manager for Windows

Universal Submit Job

Provide Network Status of Remote Windows from UNIX

Provide Network Status of Remote Windows from UNIX

This example produces a report of the network status of a remote Windows system. Instead of executing a command on the remote host, a local script file is executed.

The following figure illustrates the script file, **myscript**.

```
echo System Status
echo ----------------------------------------------------------------------------------------
date /t
time /t
echo ----------------------------------------------------------------------------------------
netstat –se
echo ----------------------------------------------------------------------------------------
netstat –a
echo ----------------------------------------------------------------------------------------
```

The following figure illustrates the command to execute the script file.

```
ucmd –script myscript -host dallas -userid joe –pwd password
```

The report is written to the standard out of UCMD.

Command Line Options

The command line options used in this example are:

Option	Description
-script	File name of a script file. The script file is sent to the remote system for execution.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

Components

Universal Command Manager for UNIX

Redirect Standard Out and Standard Error to UNIX

Redirect Standard Out and Standard Error to UNIX

The following example illustrates how to redirect the standard output and error of the DIR command to a file on the initiating system.

```
ucmd -cmd 'dir' -host dallas –userid joe –pwd password > output.file 2>&1
```

The command **dir** is sent to a remote system named **dallas** for execution. The standard output and standard error of the **dir** command are written back to the UCMD process and redirected to standard out file **output.file**. The process will authenticate and run under the authority of userid **joe**.

If the remote system is a UNIX system, change the command **dir** to **ls**.

Command Line Options

The command line options used in this example are:

Option	Description
-cmd	Remote command **ls** to execute.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

Components

Universal Command Manager for UNIX

Redirect Standard Input from Initiating System to UNIX

Redirect Standard Input from Initiating System to UNIX

The **ucmd** command reads from standard input and writes it to the UCMD Server for the remote command to read as its standard input. The allocation of standard input can be changed with a shell redirection operator. The redirection operators instruct the shell to change the allocation of the standard files. To change the allocation of standard input, use the **<** operator.

```
ucmd –script myscript -host dallas –userid joe –pwd password < input.file
```

The command is sent to a remote system named **dallas** for execution. The output of the script is redirected back to the Universal Command process’s standard out and standard error. Standard Input is read from file **input.file** on the initiating system. The process will authenticate and run under the authority of userid **joe**.

Command Line Options

Option	Description
-script	File from which to read a script file. The script file is sent to the remote system for execution.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

Components

Universal Command Manager for UNIX

Redirect Standard Input in UNIX Background Process

Redirect Standard Input in UNIX Background Process

If the command **ucmd** is executed as a background job (using the **&** operator), it will receive the SIGTTIN signal when **ucmd** tries to read from standard input. Background jobs cannot read their standard input from the terminal since the foreground job (or the shell) has it allocated. The **ucmd** job is stopped until it is brought to the foreground.

To run an **ucmd** job that does not require terminal input in the background, redirect its standard input from **/dev/null**.

```
ucmd –script myscript -host dallas –userid joe –pwd password < /dev/null &
```

The command is sent to a remote system named **dallas** for execution. The output of **myscript** is redirected back to the Universal Command process. Standard input is read from **/dev/null**. The process will authenticate and run under the authority of userid **joe**.

Command Line Options

The command line options used in this example are:

Option	Description
-script	File from which to read a script file. The script file is sent to the remote system for execution.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

Components

Universal Command Manager for Windows

Issue Universal Submit Job from UNIX to IBM i

Issue Universal Submit Job from UNIX to IBM i

The following example illustrates the issuing of a command to the remote IBM i as a parameter of the USBMJOB.

```
ucmd -cmd "usbmjob cmd(dspsyssts)" -host ohio -userid usrid -pwd usrpwd
```

In this example, USBMJOB is submitted to the server running on the host **ohio**.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-cmd	Universal Command command option (**usbmjob**).
-host	Directs the command to a computer with a host name of **ohio**.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the remote user ID.

Components

Universal Command Manager for UNIX

Universal Submit Job

Provide Network Status of Remote Windows from IBM i

Provide Network Status of Remote Windows from IBM i

This example produces a report of the network status of a remote Windows system. Instead of executing a command on the remote host, a local script file is executed.

The following figure illustrates the script file, **MYSCRIPT**.

```
echo System Status
echo ----------------------------------------------------------------------------------------
date /t
time /t
echo ----------------------------------------------------------------------------------------
netstat –se
echo ----------------------------------------------------------------------------------------
netstat –a
echo ----------------------------------------------------------------------------------------
```

The following figure illustrates the command to execute the script file.

```
STRUCM SCRIPT(myscript) HOST(dallas) USERID(joe) PWD(password)
```

The report is written to the stdout of **STRUCM**.

Command Line Options

The command line options used in this example are:

Option	Description
SCRIPT	File name of a script file. The script file is sent to the remote system for execution.
HOST	Directs the command to a computer with a host name of **dallas**.
USERID	Remote user ID with which to execute the command.
PWD	Password for the user ID.

Components

Universal Command Manager for IBM i

Execute Script to Provide Network Status of Remote Windows from IBM i

Execute Script to Provide Network Status of Remote Windows from IBM i

The following example illustrates the execution of a network status script on a remote Windows server.

```
STRUCM SCRIPT(myscript) HOST(dallas) USERID(joe) PWD(password)
```

The command **myscript** is sent to a remote system named **dallas** for execution. The standard output and standard error of **myscript** command are available to the initiating process as file **QPRINT**.

Command Line Options

Option	Description
SCRIPT	File name of a script file. The script file is sent to the remote system for execution.
HOST	Directs the command to a computer with a host name of **dallas**.
USERID	Remote user ID with which to execute the command.
PWD	Password for the user ID.

Components

Universal Command Manager for IBM i

Display Library with Manager Fault Tolerance Active Using USBMJOB

Display Library with Manager Fault Tolerance Active Using USBMJOB

The following example illustrates the use of an IBM i command on a remote system with spooling enabled.

It assumes that manager fault tolerance is active on the client platform via the UCMD configuration file. The example should execute from either a UNIX shell or a Windows system environment. The command is submitted via **USBMJOB** to allow the output data and the job log of the executed command to be brought back to the system initiating the command.

Windows	`ucmd -cmd "USBMJOB CMD(dsplib qgpl)" -userid userId -pwd password -host sysName -cmdid NTSysTest out400.txt 2>err400.txt`
UNIX	`ucmd -cmd "USBMJOB CMD(dsplib qgpl)" -userid userId -pwd userPW -host sysName -cmdid UNIX001>out400.txt 2>err400.txt`

For this example, **USBMJOB** requires no input; however, the user must supply **<NUL** to satisfy Windows operating system requirements. Without **<NUL**, the request will hang.

USBMJOB outputs data via the standard output file stream (stdout) and outputs job logs and error messages via the standard error file stream (stderr). The system takes data sent back to UCMD and stores it in **out400.txt**; it takes any error messages and the job logs and stores them in **err400.txt**.

With the Universal Command server JOBLOG_COPY_KEEP configuration option set to **yes**, a copy of the job log remains on the originating IBM i system.

The command **USBMJOB** is installed as part of UCMD Server on the IBM i system.

Command Line Options

Option	**Description**
-cmd	Remote command to execute on the IBM i.
-host	Directs the command to a computer with a host name of **sysName**.
-userid	IBM i user ID with which to execute the command.
-pwd	Password for the user ID.
-cmdid	UCMD Server (running under UBroker) puts the Command ID in its database to keep track of requests regarding a specific unit of work.

Include the -managerft command option set to **yes**, requesting manager fault tolerance, if it is not enabled via the UCMD configuration file.

Components

Universal Command Manager for IBM i

Universal Submit Job from zOS to IBM i

Universal Submit Job from z/OS to IBM i

The following figure illustrates the issuing of a command to the remote IBM i as a parameter of the USBMJOB.

```
//S1 EXEC UCMDPRC
//UNVOUT   DD SYSOUT=*
//UNVERR   DD SYSOUT=*
//SCRIPT   DD *
  ADDLIBLE lib(UNVPRD510)
  UNVPRD510/USBMJOB CMD(dsplib tuser1)
//SYSIN   DD *
  -script SCRIPT
  -host as400 -userid tuser1 -pwd tuser1
/*
```

This Universal Command Manager executes the script to a host called **as400**. UserID of **tuser1** and password of **tuser1** are used for authentication.

The script runs with the authority of UserID **tuser1**. The first line of the script adds the library **UNVPRD510** to the library concatenation of user **tuser1**. The second line executes the command **dsplib tuser1** with the USBMJOB utility.

All output created by the command will be spooled to stdout of the manager job.

USBMJOB Options

The USBMJOB option used in this example is:

Option	Description
CMD	Command that runs in the submitted batch job. The command can be a maximum of 3000 characters.

SYSIN Options

Option	Description
-script	DD from which to read a script file. The script file is sent to the remote system for execution.
-host	Directs the command to a computer with a host name of **as400**.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the remote user ID.

Components

Universal Command Manager for IBM i

Universal Submit Job

Provide Network Status of Remote Windows from HP NonStop

Provide Network Status of Remote Windows from HP NonStop

This example produces a report of the network status of a remote Windows system. Instead of executing a command on the remote host, a local script file is executed.

The following figure illustrates the script file, **myscript**.

```
echo System Status
echo ----------------------------------------------------------------------------------------
date /t
time /t
echo ----------------------------------------------------------------------------------------
netstat –se
echo ----------------------------------------------------------------------------------------
netstat –a
echo ----------------------------------------------------------------------------------------
```

The following figure illustrates the command to execute the script file.

```
run ucmd –script myscript -host dallas -userid joe –pwd password
```

The report is written to the standard out of UCMD.

Command Line Options

The command line options used in this example are:

Option	Description
-script	File name of a script file. The script file is sent to the remote system for execution.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

Components

Universal Command Manager for HP NonStop

Execute Script to Provide Network Status of Remote Windows from HP NonStop

Execute Script to Provide Network Status of Remote Windows from HP NonStop

This example executes a network status script on a remote Windows server.

```
run $SYSTEM.UNVBIN.ucmd –script myscript -host dallas -userid joe –pwd password
```

The command **myscript** is sent to a remote system named **dallas** for execution. The standard output and standard error of **myscript** command are available to the standard out of the initiating process. The process will authenticate and run under the authority of userid **joe**.

Command Line Options

The command line options used in this example are:

Option	Description
-cmd	Remote command **ucopy file** to execute.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

Components

Universal Command Manager for HP NonStop

Indesca - Remote Execution for SAP Systems

Remote Execution for SAP Systems - Overview

Overview

These pages provide information on the Remote Processing for SAP Systems feature of Indesca.

Remote Execution for SAP Systems refers to the initiation of work within an SAP system from some location outside of the SAP system. The type of work initiated within the SAP system is primarily centered on job control. Job control refers to the scheduling, running, monitoring, and managing of jobs and job data.

The Universal Connector *for Use with SAP® ERP* component of Indesca is used to execute this work on the remote SAP system.

Universal Connector operates as a single Indesca component on the local system. It accepts work requests on the local system and communicates directly with the SAP system to carry out those requests (no Indesca components are required on the SAP system).

Each work request requires a user identifier. The supplied user identifier is authenticated on the SAP system before the work can begin. If authentication is successful, the work will be performed on the SAP system under the context of the authenticated user. If authentication fails, no work is performed and the request fails.

Universal Connector communicates with SAP systems using SAP's RFC communication protocol. Work requests within the SAP system are made through external interfaces exposed by the SAP system. The primary SAP interface used by Universal Connector is XBP (eXternal Background Processing).

Work Requests

The following list identifies general work requests that can be performed on the SAP system using Indesca:

- Define/submit SAP jobs
- Modify SAP jobs
- Start SAP jobs
- Monitor SAP jobs
- Cancel running SAP jobs
- Retrieve the job log of SAP jobs
- Retrieve the spool lists of SAP jobs
- Delete SAP jobs and their associated output
- Query jobs in the SAP system
- Define/create SAP variants
- Modify SAP variants
- Query variants in the SAP system
- Raise SAP events
- Process/monitor Batch Input sessions
- Initiate/monitor Mass Activities
- Retrieve the SAP system log
- Retrieve output device information

Remote Execution for SAP Systems 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 Indesca. The examples demonstrate the use of Universal Connector to define SAP jobs.

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

Mass Activities Support in Universal Connector

Mass Activities Support in Universal Connector

Universal Connector *for Use with SAP® ERP* supports the submission, starting, and monitoring of mass activities on the SAP system.

To work with mass activities on the SAP system, Universal Connector utilizes the following SAP ABAP programs:

- **FKJO_SCHEDULE**
- **RFKK_MA_SCHEDULER**
- **RFKK_MASS_ACT_PARAMETER**

Mass Activities Process Flow

The basic process flow in working with mass activities is:

1. Create a template parameter record for the mass activity.
2. Copy the template parameter record and assign a Date ID and Run ID.
3. Schedule and start the mass activity.
4. Monitor the mass activity to completion.

The original template parameter records must be created on the SAP system using the dialogs for the given mass activity type. However, after a set of template parameter records have been created, Universal Connector can use the ABAP programs mentioned above to initiate and control the characteristics of mass activity work.

Initiating Mass Activities

Mass activities are initiated from Universal Connector by submitting and starting ABAP program **FKJO_SCHEDULE** or **RFKK_MA_SCHEDULER** . This can be accomplished by following the same procedure that would be used to submit and start any other ABAP program with Universal Connector.

For more information on submitting and starting jobs with Universal Connector, see the SUBMIT, START, and RUN commands.

Both **FKJO_SCHEDULE** and **RFKK_MA_SCHEDULER** can be used to initiate mass activities. Each program has a different approach (and different requirements) for preparing a mass activity on the SAP system. The decision of which one to use must be made by understanding the capabilities and requirements of each program and matching those to the requirements of the situation.

A discussion of the details of **FKJO_SCHEDULE** and **RFKK_MA_SCHEDULER** is beyond the scope of this document. For more information, please refer to the SAP documentation for these two programs.

The behavior of both **FKJO_SCHEDULE** and **RFKK_MA_SCHEDULER** are controlled by a set of parameters, called a variant, that apply to a specific ABAP program. Variants reside on the SAP system.

To achieve the desired results on a mass activity run, it may be necessary to modify the values of the variant used by the initiator program. In this case, initiating a mass activity becomes a two-step process:

1. Universal Connector is used to create or modify an existing variant on the SAP system.
2. Universal Connector is used to submit and start the initiator program that uses the variant.

For additional information on working with variants, see the SUBMIT VARIANT and MODIFY VARIANT commands.

Monitoring Mass Activities

Regardless of which program is used to initiate a mass activity, Universal Connector follows the same procedure for monitoring the process to completion. The MASS_ACTIVITY_WAIT command is used to instruct Universal Connector that it should perform this monitoring function (see the MASS_ACTIVITY_WAIT option).

Specifying the MASS_ACTIVITY_WAIT option will cause Universal Connector to monitor the status of the submitted / started job. In addition, as the jobs that make up the mass activity are created on the SAP system, Universal Connector detects them as child jobs of the initiator job and will begin to monitor their status as well. Universal Connector will continue to monitor the status of parent and child jobs until all jobs have completed.

Upon detecting the completion of a job, Universal Connector will optionally return the following information:

- Job log: see RETURN_JOB_LOG option.
- Application log (if one exists): see RETURN_APPLICATION_LOG option.
- Application return codes (if they were set): see RETURN_APPLICATION_RC option.
- Spooled output created by the job: see RETURN_SPOOL_LIST option.

In addition, Universal Connector will record the application return codes (if they are set) and merge them into its exit code mapping process that takes place upon program completion. Universal Connector will exit with the highest value used in the exit code processing.

Working with Parameter Records

With each mass activity run, there may be the need for parameter set adjustment.

In some cases, the ABAP program used to initiate the mass activity can perform the necessary parameter adjustments. When more detailed parameter adjustments are required, the ABAP program **RFKK_MASS_ACT_PARAMETER** can be used. In this case, Universal Connector can be used to run **RFKK_MASS_ACT_PARAMETER** by following the same procedures that would be used to run any other ABAP program on the SAP system.

For more information, see the SUBMIT, START, RUN, and WAIT commands.

The information that controls how **RFKK_MASS_ACT_PARAMETER** will adjust the mass activity parameter set is contained in a variant that resides on the SAP system. In many cases, it may be necessary to create or modify the contents of a variant with information that pertains to a specific mass activity. In this case, Universal Connector can be used to create or modify the variants as needed.

For additional information on working with variants, see the SUBMIT VARIANT and MODIFY VARIANT commands.

Mass Activities Support Example - zOS

Universal Connector Mass Activity Support Example

This example uses the Create Account Statements application to demonstrate the process of setting up a mass activity for automation with Universal Connector *for Use with SAP® ERP*.

Step 1	Via SAP GUI, create a template parameter set for a mass run: 1. In the SAP Front end GUI, enter transaction **FPCC0002** to bring up a dialog for the Create Account Statements application. 2. In the Run Identification section, enter values for Date ID and Identification. These values will be used to uniquely identify the parameter set that you create in this step. For example: • Date ID: 13.07.2010 • Identification: SBX1 3. Set up the rest of the application run parameters to meet your needs. These additional settings are not important to the concepts of this example. 4. Select Program Run->Save (or <Ctrl+S>) to save the parameter set.
Step 2	Create a variant for ABAP program RFKK_MA_SCHEDULER that will use the parameter set created in **Step 1**. This can be accomplished by running the sample job (USPVRMS) illustrated in the following figure. (The procedure for executing this JCL (USPJRMS) is illustrated in **Step 3**.) ***USPVRMS - JCL***

```
//USPVRMS  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*****************************************************************
//* Description
//* -----------
//* This sample creates a new variant named "SBX1" for ABAB program
//* RFKK_MA_SCHEDULER.
//*
//* The new variant will be set up to target Mass Activity Type
//* 0002 (Create Account Statements) using a parameter set with run
//* identification:
//* Date ID:        2010.07.15
//* Identification: SBX1
//*
//         JCLLIB ORDER=\#SHLQ.UNV.SUNVSAMP
//*
//STEP1    EXEC USPPRC
//VARDEF   DD  *
 /* Variant Header statement. */
 VARIANT\_NAME   = "SBX1"
    REPORT      = "RFKK\_MA\_SCHEDULER";

 /* Variant text statement. */
 VARIANT_TEXT   = "SBX1"
    LANGUAGE    = "EN";

 /* Mass activity type */
 SELNAME        = "P_AKTYP"
    KIND        = "P"
    LOW         = "0002";

 /* Date ID */
 SELNAME        = "P_COPYD"
    KIND        = "P"
    LOW         = "20100715";

 /* Identification */
 SELNAME        = "P_COPYI"
    KIND        = "P"
    LOW         = "SBX1";

 /* Date of Dunning Proposal Run */
 SELNAME        = "P_MAHND"
    KIND        = "P"
    LOW         = "00000000";

 /* ID of Dunning Proposal Run */
 SELNAME        = "P\_MAHNI"
    KIND        = "P"
    LOW         = "";

 /* Status for Error Messages */
 SELNAME        = "P\_STATUS"
    KIND        = "P"
    LOW         = "W";

 /* WF\_OKEY */
 SELNAME        = "WF_OKEY"
    KIND        = "P"
    LOW         = "";

 /* WF\_WITEM */
;
 SELNAME        = "WF_WITEM"
    KIND        = "P"
    LOW         = "";

 /* WF\_WLIST */
 SELNAME        = "WF_WLIST"
    KIND        = "P"
    LOW         = "";
//SYSIN    DD  *
   -dest   CF5
   -client 800
   -userid sapuid
   -pwd    sappwd
   -sub    VARDEF
/*
```

See SYSIN Options for USPVRMS JCL, below, for a description of the configuration options used in this JCL.

Step 3 Via Universal Connector, run ABAP program **RFKK_MA_SCHEDULER**; submit a Universal Connector job to initiate, monitor, and return output from the mass activity. The sample job illustrated in the following figure will accomplish this using the variant created in **Step 2**. USPPRC - JCL Procedure, shown below this JCL, is the procedure used to execute the JCL.

USPJRMS - JCL

```
//USPVRMS    JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*****************************************************************
//* Description
//* -----------
//* This sample will:
//* 1. Initiate a mass activity (via ABAP RFKK_MA_SCHEDULER).
//* 2. Monitor the mass activity (including interval jobs) to
//*    completion.
//* 3. Return output from the initiator job and all interval jobs.
//*
//* The parameter set used for the mass activity is specified in
//* the variant passed to RFKK_MA_SCHEDULER.  In this case, we are
//* using variant SBX1.
//*
//         JCLLIB ORDER=#SHLQ.UNV.SUNVSAMP
//*
//STEP1    EXEC USPPRC
//JOBDEF   DD  *
 /* Job Header statement. */
 JOBNAME = "RFKK_MA_SCHEDULER_SBX1";

 /* ABAP Step statement. */
 ABAP_STEP            = "STEP 1"
    ABAP_PROGRAM_NAME = "RFKK_MA_SCHEDULER"
    VARIANT_NAME      = "SBX1";
//SYSIN    DD  *
   -dest   CF5
   -client 800
   -userid sapuid
   -pwd    sappwd
   -sub    VARDEF
   -start
   -mawait
/*
```

See SYSIN Options for USPJRMS JCL, below, for a description of the configuration options used in this JCL.

USPPRC - JCL Procedure

```
//USPPRC   PROC UPARM=,              -- USAP options
//             SAPRFC=USPRFC00,      -- SAP RFC member
//             USAPPRE=#SHLQ.UNV,
//             USAPPRD=#PHLQ.UNV
//*
//PS1      EXEC PGM=USAP,PARM='ENVAR(TZ=EST5EDT)/&UPARM'
//STEPLIB  DD  DISP=SHR,DSN=&USAPPRE..SUNVLOAD
//*
//UNVNLS   DD  DISP=SHR,DSN=&USAPPRE..SUNVNLS
//UNVRFC   DD  DISP=SHR,DSN=&USAPPRD..UNVCONF(&SAPRFC)
//UNVTRACE DD  SYSOUT=*
//*
//SYSPRINT DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
```

SYSIN Options for USPVRMS JCL

The SYSIN options used in the USPVRMS JCL example are:

Option	Description
-dest	Name of a destination defined in the saprfc.ini file.
-client	SAP client number.
-userid	SAP user ID with which to logon to the SAP system.
-pwd	Password for the SAP user ID.
-sub	Definition of the job to the SAP system.

SYSIN Options for USPJRMS JCL

The SYSIN options used in the USPJRMS JCL example are:

Option	Description
-dest	Name of a destination defined in the saprfc.ini file.
-client	SAP client number.
-userid	SAP user ID with which to logon to the SAP system.
-pwd	Password for the SAP user ID.
-sub	Definition of the job to the SAP system.
-start	Starts the newly defined job.
-mawait	Causes USAP to wait for the SAP mass activity jobs to complete processing.

Components

Universal Connector for z/OS

Batch Input Monitoring in Universal Connector

Overview

Universal Connector *for Use with SAP® ERP* supports the monitoring of batch input session processing. This support is currently limited to SAP 4.6C and above. To perform batch input monitoring, Universal Connector utilizes the functionality of SAP's ABAP program **RSBDCSUB**.

RSBDCSUB selects batch input sessions for processing based on the criteria specified in its variant. The batch input sessions selected to be processed by **RSBDCSUB** are transferred to the SAP system's background processing. **RSBDCSUB** completes independent of the session processing jobs it starts.

The spoollist produced by **RSBDCSUB** contains the information required to identify the session processing jobs created, and relate them to their respective batch input sessions. This information consists of a job name (same as session name), job id, and queue id. The job name / job id combination uniquely identifies the session processing job. The queue id uniquely identifies the queue that contains the batch input session data and status.

Batch Input Monitoring Process

The following steps illustrate the basic overview of the Universal Connector batch input monitoring process:

Step 1	Universal Connector starts a single step job that executes ABAP program **RSBDCSUB**, or USAP connects to a previously started single step job executing ABAP program **RSBDCSUB**.
Step 2	Universal Connector waits for the **RSBDCSUB** job to complete.
Step 3	If the **RSBDCSUB** job terminates, Universal Connector exits with the Universal Connector 'Terminated' job status code. Otherwise, Universal Connector retrieves the spoollist generated by **RSBDCSUB** and extracts the session processing information. This information consists of the session processing jobs that were kicked off by **RSBDCSUB**, and the corresponding queues that contain the sessions.
Step 4	Universal Connector begins to monitor all session processing jobs that were kicked off by **RSBDCSUB**. When Universal Connector detects that a session processing job has completed, it retrieves the state of the corresponding queue and converts the queue state to a Universal Connector queue state exit code. Universal Connector continues this monitoring process until all session processing jobs have completed.
Step 5	When all session processing jobs have completed, Universal Connector exits with the highest queue state exit code retrieved from all sessions that were processed by **RSBDCSUB**.

Batch Input Monitoring Requirements

SAP System

Universal Connector only supports batch input monitoring on SAP 4.6 systems. This restriction is based on the ABAP program **RSBDCSUB**. **RSBDCSUB** initiates session processing jobs and completes independent of the session processing jobs.

Only the SAP 4.6 version of **RSBDCSUB** produces a spoollist that contains all the information needed to monitor the session processing jobs and the states of the sessions they process. This information consists of the job name and job id of the session processing jobs that get initiated, and the queue id of the session that is being processed.

SAP Batch Input Sessions

All batch input sessions that will be monitored by Universal Connector must have the **keep session** flag checked. This is required because the queue that contains the batch input session must exist in the SAP system after the session processing job completes in order for Universal Connector to retrieve the state of the queue.

Universal Connector

To perform batch input monitoring with Universal Connector, a single step SAP job must be started that executes ABAP program **RSBDCSUB**. Universal Connector can start the job or can connect to a job that was previously started.

Universal Connector uses the spoollist generated by **RSBDCSUB** to extract session processing information. The format of this report depends on the language of the job step. There are three Universal Connector parameters that must be set up for the language being used. See BDCWAIT - USAP Command. By default, these parameters are set up to work with the English language.

The print parameters for the job step executing **RSBDCSUB** must specify enough columns to allow the full width of the report to be generated without truncation. A value of 132 is sufficient. In addition, the number of lines per page must allow the entire report to be generated on a single page. This is due to limitations in the **RSBDCSUB** report generation capability.

The Universal Connector command line parameter **-bdcwait** is used to initiate the batch input monitoring process. For details on this parameter, see BDCWAIT - USAP Command.

Batch Input Monitoring Parameters

The set of Universal Connector configuration parameters that are specific to the batch input monitoring support are:

- BDC Wait
- BDC Job Name Pattern
- BDC Job ID Pattern
- BDC Queue ID Pattern
- Queue **to be created** exit code mapping
- Queue **unprocessed** exit code mapping
- Queue **in background** exit code mapping
- Queue **finished** exit code mapping
- Queue **error** exit code mapping

See BDCWAIT - USAP Command for details concerning the use of these parameters.

Batch Input Monitoring Example - zOS

Batch Input Processing Example

This example illustrates batch input processing for z/OS.

```
//USPBDC1  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*****************************************************************
//* Description
//* -----------
//* This sample demonstrates the use of USAP's Batch Input
//* Monitoring.
//*
//* NOTE: This job requires that variant SBX1 exists for ABAP
//* program RSBDCSUB.
//*
//* This sample will:
//* 1. Modify variant SBX1 with values required for this
//*    job run (specifies the batch input session to be processed).
//* 2. Submit a new job to the SAP system.
//* 3. Start the job.
//* 4. Monitor the submitted job and all session processing jobs
//*    to completion.
//* 5. Return the job logs.
//* 6. Return the spool list.
//* 7. Prints a brief report indicating the status of all batch
//*    input sessions processed
//*
//         JCLLIB ORDER=\#SHLQ.UNV.SUNVSAMP
//*
//*****************************************************************
//* Modify variant 'SBX1' for ABAP program RSBDCSUB
//*****************************************************************
//STEP1    EXEC USPPRC
//VARDEF   DD  *
 /* Variant Header statement. */
 VARIANT\_NAME   = "SBX1"
    REPORT      = "RSBDCSUB";
 /* Session */
 SELNAME        = "MAPPE"
    KIND        = "P"
    SIGN        = ""
    OPTION      = ""
    LOW         = "SBX20100720"
    HIGH        = "";
//SYSIN    DD  *
   -dest         CF5
   -client       800
   -userid       sapuid
   -pwd          sappwd
   -modify       VARDEF
/*
//*****************************************************************
//* Run ABAP program RSBDCSUB to perform Batch Input processing
//* using the variant that was modified in step 1.
//*
//* NOTE: This job requires that a variant SBX1 exists for ABAP
//* program RSBDCSUB.
//*
//*****************************************************************
//STEP2    EXEC USPPRC
//JOBDEF   DD  *
 /* Job Header statement. */
 JOBNAME = "RSBDCSUB";
 /* ABAP Step statement. */
 ABAP_STEP            = "STEP 1"
    ABAP_PROGRAM_NAME = "RSBDCSUB_SBX1"
    VARIANT_NAME      = "SBX1";
//SYSIN    DD  *
   -dest    CF5
   -client  800
   -userid  sapuid
   -pwd     sappwd
   -sub     JOBDEF
   -start
   -bdcwait
/*
```

SYSIN Options

SYSIN options used in this example are:

Command Options	Description
-dest	Named set of connection parameters (destination) 'CF5'. These connection parameters are used for communications with the SAP system. The default file for destination parameters is **#HLQ.UNV.USPRFC00** .
-client	SAP client number that the Universal Connector will communicate with.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-sub	Specification that Universal Connector will issue the SUBMIT command.
-start	Specification that Universal Connector will instruct the SAP system to start the submitted job.

Components

Universal Connector for z/OS

Remote Execution for SAP Systems - Examples

Remote Execution for SAP Systems Examples - z/OS and UNIX

- Define Job, Run Job, Get Output, and Purge Job

Remote Execution for SAP Systems Examples - z/OS

- Submitting Job to SAP Using SAP Job as Template
- Submitting Job to SAP Using Job Definition File
- Running Job on SAP Using SAP Job
- Running Job on SAP Using Job Definition File
- Running an SAP Job on a Specific SAP Server
- Variant Substitution
- Creating a Variant Substitution Using GENERATE VARDEF Command
- Creating a Job Definition Using GENERATE JOBDEF Command

Remote Execution for SAP Systems Examples - UNIX

- Submitting an SAP Job Using SAP Job as Template
- Submitting an SAP Job Using Job Definition File
- Running an SAP Job Using SAP Job as Template
- Running an SAP Job Using a Job Definition File
- Running an SAP Job on a Specific SAP Server
- Variant Substitution
- Creating a Variant Definition Using GENERATE VARDEF Command
- Creating Job Definition Using GENERATE JOBDEF Command

Define Job, Run Job, Get Output, and Purge Job

Define Job, Run Job, Get Output, and Purge Job

This example uses an existing job in an SAP system as a model and creates a copy.

The newly created job then is started. Universal Connector waits for the job to finish, and then writes the joblog to standard error and the spoollists to standard out.

Finally, the job and its output are purged from the SAP system.

```
usap –sub –j SAMPLE1 –b 10080901 –start –wait –purge –userid sapuser –pwd sappwd –dest BIN_HS0092
–client 800
```

Command Options

The command options used are:

Command Options	Description
-sub	Submit command which defines a job to an SAP system. The lack of a job definition file indicates that the definition will use an existing job as a model. That job will be identified by -jobname and -jobid.
-start	Specification that the job should be started.
-wait	Causes Universal Connector to wait for the job to complete.
-purge	Specification that the job and its associated output are to be purged from the SAP system.
-userid	External SAP user ID with which the command is executed.
-pwd	Password for the user ID.
-dest	Destination name in the **saprfc.ini** file.
-client	SAP client number.

Components

Universal Connector for z/OS

Universal Connector for UNIX

Submitting Job to SAP Using SAP Job as Template - zOS

Submitting a Job to an SAP System Using a Pre-existing SAP Job as a Template – z/OS

This example illustrates submitting a job to an SAP system using a pre-existing SAP job as a template for the submitted job.

```
//USPSUB1  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*****************************************************************
//* Description
//* -----------
//* This sample will submit a new job to an SAP system using a
//* pre-existing SAP job as a template.
//*
//* NOTE: This job assumes (and requires) that a job already
//* exists on an the SAP system with:
//* Job Name: USPSUB1
//* Job ID:   12345678
//*
//* After running this job, a new SAP job will be created on the
//* SAP system.  The new job will have the same name as the
//* pre-existing job that was used as a template. However, the
//* SAP system will assign a new job ID.
//*
//         JCLLIB ORDER=#SHLQ.UNV.SUNVSAMP
//*
//STEP1    EXEC USPPRC
//SYSIN    DD  *
   -dest      CF5
   -client    800
   -userid    sapuid
   -pwd       sappwd
   -sub
   -jobname   USPSUB1
   -jobid     12345678
/*
```

The JCL procedure USPPRC is used to execute the Universal Connector command. Universal Connector connects to the SAP system and performs the requested work. In this case, a new job is created on the SAP system that is identical to the template job with the exception of job ID.

SYSIN Options

SYSIN options used in this example are:

Command Options	Description
-dest	Named set of connection parameters (destination) 'CF5'. These connection parameters are used for communications with the SAP system. The default file for destination parameters is **#HLQ.UNV.USPRFC00** .
-client	SAP client number that the Universal Connector will communicate with.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

-sub	Specification that Universal Connector will issue the SUBMIT command.
-jobname	Job name of the SAP job that will be used as a template.
-jobid	Job ID of the SAP job that will be used as a template.

Components

Universal Connector for z/OS

Submitting Job to SAP Using Job Definition File - zOS

Submitting a Job to an SAP System Using a Universal Connector Job Definition File – z/OS

This example illustrates submitting a job to an SAP system using Universal Connector job definition file.

```
//USPSUB2  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*****************************************************************
//* Description
//* -----------
//* This sample will submit a new job to an SAP system.
//*
//         JCLLIB ORDER=#SHLQ.UNV.SUNVSAMP
//*
//STEP1    EXEC USPPRC
//JOBDEF   DD  *
 /* Job Header statement. */
 JOBNAME = "USPSUB2";

 /* ABAP Step statement. */
 ABAP_STEP            = "STEP 1"
    ABAP_PROGRAM_NAME = "BTCSPOOL";
//SYSIN    DD  *
   -dest   CF5
   -client 800
   -userid sapuid
   -pwd    sappwd
   -sub    JOBDEF
/*
```

The JCL procedure USPPRC is used to execute the Universal Connector command. Universal Connector connects to the SAP system and performs the requested work. In this case, a new job is created on the SAP system based on a definition that was provided in a Universal Connector definition file.

SYSIN Options

SYSIN options used in this example are:

Command Options	Description
-dest	Named set of connection parameters (destination) 'CF5'. These connection parameters are used for communications with the SAP system. The default file for destination parameters is **#HLQ.UNV.USPRFC00** .
-client	SAP client number that the Universal Connector will communicate with.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-sub	Specification that Universal Connector will issue the SUBMIT command.

Components

Universal Connector for z/OS

Running Job on SAP Using SAP Job - zOS

Running a Job on an SAP System Using a Pre-existing SAP Job – z/OS

This example illustrates running a job on an SAP system using a pre-existing SAP job.

```
//USPRUN1  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*****************************************************************
//* Description
//* -----------
//* This sample will:
//* 1. Submit a new job to an SAP system using a pre-existing SAP
//*    job as a template.
//* 2. Start the newly created job.
//* 3. Wait for the job to complete.
//* 4. Return the job log.
//* 5. Return the spool list.
//* 6. The SAP job completion status will be mapped to an exit
//*    code.  USAP will exit with the mapped exit code.
//*
//* NOTE: This job assumes (and requires) that a job already
//* exists on the SAP system with:
//* Job Name: USPRUN1
//* Job ID:   12345678
//*
//         JCLLIB ORDER=#SHLQ.UNV.SUNVSAMP
//*
//STEP1    EXEC USPPRC
//SYSIN    DD  *
   -dest      CF5
   -client    800
   -userid    sapuid
   -pwd       sappwd
   -sub
   -jobname   USPRUN1
   -jobid     12345678
   -start
   -wait
   -joblog    yes
   -spoollist yes
/*
```

SYSIN Options

SYSIN options used in this example are:

Command Options	Description
-dest	Named set of connection parameters (destination) 'CF5'. These connection parameters are used for communications with the SAP system. The default file for destination parameters is **#HLQ.UNV.USPRFC00** .
-client	SAP client number that the Universal Connector will communicate with.
-userid	Remote user ID with which to execute the command.

-pwd	Password for the user ID.
-sub	Specification that Universal Connector will issue the SUBMIT command.
-jobname	Job name of the SAP job that will be used as a template.
-jobid	Job ID of the SAP job that will be used as a template.
-start	Specification that Universal Connector will instruct the SAP system to start the submitted job.
-wait	Specification that Universal Connector will monitor the started job until it completes.
-joblog	Specification that Universal Connector will return the SAP log for the started job.
-spoollist	Specification that Universal Connector will return any spool lists created by the started job.

Components

Universal Connector for z/OS

Running Job on SAP Using Job Definition File - zOS

Running a Job on an SAP System Using a Universal Connector Job Definition File – z/OS

This example illustrates running a job on an SAP system using a Universal Connector job definition file.

```
//USPRUN2  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*****************************************************************
//* Description
//* -----------
//* This sample will:
//* 1. Submit a new job to an SAP system.
//* 2. Start the job.
//* 3. Wait for the job to complete.
//* 4. Return the job log.
//* 5. Return the spool list.
//*
//         JCLLIB ORDER=#SHLQ.UNV.SUNVSAMP
//*
//STEP1    EXEC USPPRC
//JOBDEF   DD  *
 /* Job Header statement. */
 JOBNAME = "USPRUN";

 /* ABAP Step statement. */
 ABAP_STEP            = "STEP 1"
    ABAP_PROGRAM_NAME = "BTCSPOOL";
//SYSIN    DD  *
   -dest         CF5
   -client       800
   -userid       sapuid
   -pwd          sappwd
   -sub          JOBDEF
   -start
   -wait
   -joblog       yes
   -spoollist    yes
/*
```

SYSIN Options

SYSIN options used in this example are:

Command Options	Description
-dest	Named set of connection parameters (destination) 'CF5'. These connection parameters are used for communications with the SAP system. The default file for destination parameters is **#HLQ.UNV.USPRFC00** .
-client	SAP client number that Universal Connector will communicate with.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

-sub	Specification that Universal Connector will issue the SUBMIT command.
-start	Specification that Universal Connector will instruct the SAP system to start the submitted job.
-wait	Specification that Universal Connector will monitor the started job until it completes.
-joblog	Specification that Universal Connector will return the SAP log for the started job.
-spoollist	Specification that Universal Connector will return any spool lists created by the started job.

Components

Universal Connector for z/OS

Running an SAP Job on a Specific SAP Server - zOS

Running an SAP Job on a Specific SAP Server – z/OS

This example illustrates running an SAP job on a specific SAP Server.

```
//USPRUN3  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*****************************************************************
//* Description
//* -----------
//* This sample demonstrates how to specify a specific SAP server
//* for the SAP job to run on.
//*
//* This sample will:
//* 1. Submit a new job to an SAP system.
//* 2. Start the job on a specific SAP server.
//* 3. Wait for the job to complete.
//* 4. Return the job log.
//* 5. Return the spool list.
//*
//         JCLLIB ORDER=#SHLQ.UNV.SUNVSAMP
//*
//STEP1    EXEC USPPRC
//JOBDEF   DD  *
 /* Job Header statement. */
 JOBNAME = "USPRUN3";

 /* ABAP Step statement. */
 ABAP_STEP            = "STEP 1"
    ABAP_PROGRAM_NAME = "BTCSPOOL";
//SYSIN    DD  *
   -dest         CF5
   -client       800
   -userid       sapuid
   -pwd          sappwd
   -sub          JOBDEF
   -start
   -targetserver pwdf2643
   -wait
   -joblog       yes
   -spoollist    yes
 /*
```

SYSIN Options

SYSIN options used in this example are:

Command Options	Description
-dest	Named set of connection parameters (destination) 'CF5'. These connection parameters are used for communications with the SAP system. The default file for destination parameters is **#HLQ.UNV.USPRFC00** .
-client	SAP client number that the Universal Connector will communicate with.
-userid	Remote user ID with which to execute the command.

-pwd	Password for the user ID.
-sub	Specification that Universal Connector will issue the SUBMIT command.
-start	Specification that Universal Connector will instruct the SAP system to start the submitted job.
-targetserver	Target server for the SAP job to run on.
-wait	Specification that Universal Connector will monitor the started job until it completes.
-joblog	Specification that Universal Connector will return the SAP log for the started job.
-spoollist	Specification that Universal Connector will return any spool lists created by the started job.

Components

Universal Connector for z/OS

Variant Substitution - zOS

Variant Substitution – z/OS

This example demonstrates the use of variant substitution.

When Universal Connector is using pre-defined SAP jobs as template jobs (rather than Universal Connector job definition files), it may be necessary or desirable to replace the variants specified in the template job with variants more appropriate for the current job run. In this case, Universal Connector's target_variant option can be used to accomplish the variant substitution.

```
//USPVARSB  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*****************************************************************
//* Description
//* -----------
//* This sample demonstrates the use of USAP's target_variant
//* option to perform variant substitution when using pre-defined
//* SAP jobs as templates.
//*
//* NOTE: This job assumes (and requires) that a job already
//* exists on an the SAP system with:
//* Job Name: VARSBST1
//* Job ID:   12345678
//*
//* This sample will:
//* 1. Modify variants SBT1 and SBT2 with values required for this
//*    job run.
//* 2. Submit a new job to the SAP system using a pre-existing SAP
//*    job as a template.
//* 3. Perform variant substitution on the newly created job.  The
//*    newly created job will now use variants SBT1 and SBT2 for
//*    steps 1 and 2 respectively (regardless of what variants
//*    were defined in the template job).
//* 4. Wait for the job to complete.
//* 5. Return the job log.
//* 6. Return the spool list.
//*
//         JCLLIB ORDER=#SHLQ.UNV.SUNVSAMP
//*
//*****************************************************************
//* Modify variant 'SBT1' for ABAP program RSUSR002
//*****************************************************************
//STEP1    EXEC USPPRC
//VARDEF   DD  *
 /* Variant Header statement. */
 VARIANT_NAME   = "SBT1"
    REPORT      = "RSUSR002";
 /* User */
 SELNAME        = "USER"
    KIND        = "S"
    SIGN        = "I"
    OPTION      = "CP"
    LOW         = "STONEBRANCH"
    HIGH        = "";
//SYSIN    DD  *
   -dest         CF5
   -client       800
   -userid       sapuid
   -pwd          sappwd
   -modify       VARDEF
/*
//*****************************************************************
//* Modify variant 'SBT2' for ABAP program RSUSR002
//*****************************************************************
//STEP2    EXEC USPPRC
//VARDEF   DD  *
 /* Variant Header statement. */
 VARIANT_NAME   = "SBT2"
```

```
    REPORT      = "RSUSR002";
 /* User */
 SELNAME        = "USER"
    KIND        = "S"
    SIGN        = "I"
    OPTION      = "CP"
    LOW         = "STONEBRANCH1"
    HIGH        = "";
//SYSIN    DD  *
   -dest         CF5
   -client       800
   -userid       sapuid
   -pwd          sappwd
   -modify       VARDEF
/*
//*****************************************************************
//* Run SAP job using a pre-defined SAP job as a template and
//* perform variant substitution.
//*
//* NOTE: This job assumes (and requires) that a job already
//* exists on an the SAP system with:
//* Job Name: VARSBST1
//* Job ID:   12345678
//*
//* The pre-defined job must have ABAP program RSUSR002 defined in
//* step one and step two.
//*****************************************************************
//STEP3    EXEC USPPRC
//SYSIN    DD  *
   -dest         CF5
   -client       800
   -userid       sapuid
   -pwd          sappwd
   -run
   -jobname      VARSBST1
   -jobid        12345678
```

```
    -target_variant 1,SBT1;2,SBT2
/*
```

SYSIN Options

SYSIN options used in this example are:

Command Options	Description
-dest	Named set of connection parameters (destination) 'CF5'. These connection parameters are used for communications with the SAP system. The default file for destination parameters is **#HLQ.UNV.USPRFC00** .
-client	SAP client number that the Universal Connector will communicate with.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-modify	Specification that Universal Connector will issue the modify command for the specified definition file.
-run	Specification that Universal Connector will run the new job.
-jobname	Job name of the SAP job that will be used as a template.
-jobid	Job ID of the SAP job that will be used as a template.
-target_variant	Replacement variants for the ABAP program job steps.

Components

Universal Connector for z/OS

Creating a Variant Substitution Using GENERATE VARDEF Command - zOS

Creating a Universal Connector Variant Definition Using the GENERATE VARDEF Command – z/OS

SAP variants often have many parameters. Manually creating Universal Connector variant definitions can be tedious and time consuming.

Universal Connector offers a function that generates a complete variant definition based on a pre-existing template variant on the SAP system. The generated variant definition then can be used with a Universal Connector SUBMIT or MODIFY command to prepare a variant for a job run.

The following example demonstrates the use of the GENERATE VARDEF command.

```
//USPGEN1  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*****************************************************************
//* Description
//* -----------
//* This sample generates a USAP variant definition based on a
//* pre-existing template variant on an SAP system.
//*
//* NOTE: This job assumes (and requires) that a variant named SBT1
//* exists for ABAP program RSBDCSUB.
//*
//         JCLLIB ORDER=#SHLQ.UNV.SUNVSAMP
//*
//STEP1    EXEC USPPRC
//SYSIN    DD  *
   -dest     CF5
   -client   800
   -userid   sapuid
   -pwd      sappwd
   -generate vardef
   -abapname RSBDCSUB
   -variant  SBT1
/*
```

SYSIN Options

SYSIN options used in this example are:

Command Options	Description
-dest	Named set of connection parameters (destination) 'CF5'. These connection parameters are used for communications with the SAP system. The default file for destination parameters is **#HLQ.UNV.USPRFC00** .
-client	SAP client number that the Universal Connector will communicate with.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-generate	Instructs Universal Connector to generate the specified variant definition.

-abapname	Name of the ABAP program that the template variant belongs to.
-variant	Name of the variant that Universal Connector will use as a template for generation.

Components

Universal Connector for z/OS

Creating a Job Definition Using GENERATE JOBDEF Command - zOS

Creating a Universal Connector Job Definition Using the GENERATE JOBDEF Command – z/OS

SAP jobs offer many configuration options. Manually creating Universal Connector job definitions that utilize many configuration options can be tedious and time consuming.

Universal Connector offers a function that generates a complete job definition based on a pre-existing template job on the SAP system. The generated job definition can then be modified, if needed.

The following example demonstrates the use of the generate jobdef command.

```
//USPGEN2  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*****************************************************************
//* Description
//* -----------
//* This sample generates a USAP job definition based on a
//* pre-existing template job on an SAP system.
//*
//* NOTE: This job assumes (and requires) that a job already
//* exists on an the SAP system with:
//* Job Name: USP_TEMPLATE_1
//* Job ID:   12345678
//*
//         JCLLIB ORDER=#SHLQ.UNV.SUNVSAMP
//*
//STEP1    EXEC USPPRC
//SYSIN    DD  *
-dest     CF5
-client   800
-userid   sapuid
-pwd      sappwd
-generate jobdef
-jobname  USP_TEMPLATE_1
-jobid    12345678
/*
```

SYSIN Options

SYSIN options used in this example are:

Command Options	Description
-dest	Named set of connection parameters (destination) 'CF5'. These connection parameters are used for communications with the SAP system. The default file for destination parameters is **#HLQ.UNV.USPRFC00** .
-client	SAP client number that the Universal Connector will communicate with.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-generate	Instructs Universal Connector to generate the specified variant definition.

-jobname	Name of the SAP job that will be used as a template for generation.
-jobid	Job ID of the SAP job that will be used as a template for generation.

Components

Universal Connector for z/OS

Submitting an SAP Job Using SAP Job as Template - UNIX

Submitting a Job to an SAP System Using a Pre-existing SAP Job as a Template – UNIX

This example illustrates submitting a job to an SAP system using a pre-existing SAP job as a template for the submitted job.

⚠ **Note**
This job assumes (and requires) that a job already exists on an the SAP system with:

- Job Name: **USPSUB1**
- Job ID: **12345678**

After running this job, a new SAP job will be created on the SAP system. The new job will be identical to the template job with the exception of job ID. The SAP system will assign a new job ID.

The following figure illustrates the command to submit the job.

```
usap -dest CF5 -client 800 -userid sapuid -pwd sappwd -sub
-jobname USPSUB1 -jobid 12345678
```

Command Line Options

Command line options used in this example are:

Command Options	Description
-dest	Named set of connection parameters (destination) 'CF5'. These connection parameters are used for communications with the SAP system. The "destinations" are stored in file **saprfc.ini**, which must be in the current directory, or its full path must be specified in environment variable **RFC_INI**.
-client	SAP client number that the Universal Connector will communicate with.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-sub	Specification that Universal Connector will issue the SUBMIT command.
-jobname	Job name of the SAP job that will be used as a template.
-jobid	Job ID of the SAP job that will be used as a template.

Components

Universal Connector for UNIX

Submitting an SAP Job Using Job Definition File - UNIX

Submitting a Job to an SAP System Using a Universal Connector Job Definition File – UNIX

This example illustrates submitting a job to an SAP system using Universal Connector job definition file.

The following figure illustrates the job definition file.

```
/* Job Header statement. */
 JOBNAME = "USPSUB2";

/* ABAP Step statement. */
ABAP_STEP            = "STEP 1"
ABAP_PROGRAM_NAME = "BTCSPOOL";
```

The following figure illustrates the command to submit the job.

```
usap -dest CF5 -client 800 -userid sapuid -pwd sappwd
     -sub jobdefFile
```

After running this job, a new SAP job will be created on the SAP system with job name **USPSUB2**. The job will contain one step that runs ABAP program **BTCSPOOL**.

Command Line Options

Command line options used in this example are:

Command Options	Description
-dest	Named set of connection parameters (destination) 'CF5'. These connection parameters are used for communications with the SAP system. The "destinations" are stored in file **saprfc.ini**, which must be in the current directory, or its full path must be specified in environment variable **RFC_INI**.
-client	SAP client number that the Universal Connector will communicate with.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-sub	Specification that Universal Connector will issue the SUBMIT command.

Components

Universal Connector for UNIX

Running an SAP Job Using SAP Job as Template - UNIX

Running a Job on an SAP System Using a Pre-existing SAP Job – UNIX

This example illustrates running a job on an SAP system using a pre-existing SAP job.

Executing this example will:

1	Submit a new job to an SAP system using a pre-existing SAP job as a template.
2	Start the newly created job.
3	Wait for the job to complete.
4	Return the job log.
5	Return the spool list.
6	The SAP job completion status will be mapped to an exit code and Universal Connector will exit with the mapped exit code.

Note
This job assumes (and requires) that a job already exists on an the SAP system with:

- Job Name: **USPRUN1**
- Job ID: **12345678**

The following figure illustrates the command to run the job.

```
usap -dest CF5 -client 800 -userid sapuid -pwd sappwd
     -sub -jobname USPRUN1 -jobid 12345678 -start -wait
     -joblog yes -spoollist yes
```

Command Line Options

Command line options used in this example are:

Command Options	Description
-dest	Named set of connection parameters (destination) 'CF5'. These connection parameters are used for communications with the SAP system. The "destinations" are stored in file **saprfc.ini**, which must be in the current directory, or its full path must be specified in environment variable **RFC_INI**.
-client	SAP client number that the Universal Connector will communicate with.
-userid	Remote user ID with which to execute the command.

-pwd	Password for the user ID.
-sub	Specification that Universal Connector will issue the SUBMIT command.
-jobname	Job name of the SAP job that will be used as a template.
-jobid	Job ID of the SAP job that will be used as a template.
-start	Specification that Universal Connector will instruct the SAP system to start the submitted job.
-wait	Specification that Universal Connector will monitor the started job until it completes.
-joblog	Specification that Universal Connector will return the SAP log for the started job.
-spoollist	Specification that Universal Connector will return any spool lists created by the started job.

Components

Universal Connector for UNIX

Running an SAP Job Using a Job Definition File - UNIX

Running a Job on an SAP System Using a Universal Connector Job Definition File – UNIX

This example illustrates running a job on an SAP system using a Universal Connector job definition file.

Executing this sample will:

1	Submit a new job to an SAP system.
2	Start the job.
3	Wait for the job to complete.
4	Return the job log.
5	Return the spool list.

The following figure illustrates the job definition file.

```
/* Job Header statement. */
 JOBNAME = "USPRUN";

 /* ABAP Step statement. */
 ABAP_STEP            = "STEP 1"
    ABAP_PROGRAM_NAME = "BTCSPOOL";
```

The following figure illustrates the command to run the job.

```
usap -dest CF5 -client 800 -userid sapuid -pwd sappwd
     -sub JOBDEF -start -wait -joblog yes -spoollist yes
```

Command Line Options

Command line options used in this example are:

Command Options	Description
-dest	Named set of connection parameters (destination) 'CF5'. These connection parameters are used for communications with the SAP system. The "destinations" are stored in file **saprfc.ini**, which must be in the current directory, or its full path must be specified in environment variable **RFC_INI**.
-client	SAP client number that the Universal Connector will communicate with.
-userid	Remote user ID with which to execute the command.

-pwd	Password for the user ID.
-sub	Specification that Universal Connector will issue the SUBMIT command.
-start	Specification that Universal Connector will instruct the SAP system to start the submitted job.
-wait	Specification that Universal Connector will monitor the started job until it completes.
-joblog	Specification that Universal Connector will return the SAP log for the started job.
-spoollist	Specification that Universal Connector will return any spool lists created by the started job.

Components

Universal Connector for UNIX

Running an SAP Job on a Specific SAP Server - UNIX

Running a Job on an SAP System on a Specific SAP Server – UNIX

This example illustrates running a job on an SAP system on a specific SAP Server.

Executing this example will:

1	Submit a new job to an SAP system.
2	Start the job on a specific SAP server.
3	Wait for the job to complete.
4	Return the job log.
5	Return the spool list.

The following figure illustrates the job definition file.

```
/* Job Header statement. */
 JOBNAME = "USPRUN3";

 /* ABAP Step statement. */
 ABAP_STEP             = "STEP 1"
    ABAP_PROGRAM_NAME = "BTCSPOOL";
```

The following figure illustrates the command to run the job.

```
usap -dest CF5 -client 800 -userid sapuid -pwd sappwd
     -sub jobdefFile -start -targetserver pwdf2643 -wait
     -joblog yes -spoollist yes
```

Command Line Options

Command line options used in this example are:

Command Options	Description
-dest	Named set of connection parameters (destination) 'CF5'. These connection parameters are used for communications with the SAP system. The "destinations" are stored in file **saprfc.ini**, which must be in the current directory, or its full path must be specified in environment variable **RFC_INI**.
-client	SAP client number that the Universal Connector will communicate with.
-userid	Remote user ID with which to execute the command.

-pwd	Password for the user ID.
-sub	Specification that Universal Connector will issue the SUBMIT command.
-start	Specification that Universal Connector will instruct the SAP system to start the submitted job.
-targetserver	Target server for the SAP job to run on.
-wait	Specification that Universal Connector will monitor the started job until it completes.
-joblog	Specification that Universal Connector will return the SAP log for the started job.
-spoollist	Specification that Universal Connector will return any spool lists created by the started job.

Components

Universal Connector for UNIX

Variant Substitution - UNIX

Variant Substitution – UNIX

This example demonstrates the use of variant substitution.

When Universal Connector is using pre-defined SAP jobs as template jobs (rather than USAP job definition files), it may be necessary or desirable to replace the variants specified in the template job with variants more appropriate for the current job run. In this case, Universal Connector's TARGET_VARIANT option can be used to accomplish the variant substitution.

The example is comprised of three steps:

Step 1	Modify SAP variant SBT1.
Step 2	Modify SAP variant SBT2.
Step 3	Run a new SAP job that is created using a pre-existing SAP job as a template. Variant substitution is performed on the newly created job. As a result, the newly created job will run using the variants that were modified in steps 1 and 2.

Executing this example will:

1	Modify variants SBT1 and SBT2 with values required for this job run.
2	Submit a new job to the SAP system using a pre-existing SAP job as a template.
3	Perform variant substitution on the newly created job. The newly created job will now use variants SBT1 and SBT2 for steps 1 and 2, respectively (regardless of what variants were defined in the template job).
4	Wait for the job to complete.
5	Return the job log.
6	Return the spool list.

⚠ **Note**
This job assumes (and requires) that a job already exists on an the SAP system with:

- Job Name: **VARSBST1**
- Job ID: **12345678**

Step 1

This step modifies SAP variant **SBT1**.

The following figure illustrates the variant definition file for variant **SBT1**.

```
/* Variant Header statement. */
VARIANT_NAME   = "SBT1"
   REPORT      = "RSUSR002";

/* User */
SELNAME        = "USER"
   KIND        = "S"
   SIGN        = "I"
   OPTION      = "CP"
   LOW         = "STONEBRANCH"
   HIGH        = "";
```

The following figure illustrates the command line to modify variant **SBT1**.

```
usap -dest CF5 -client 800 -userid sapuid -pwd sappwd
     -modify vardefFile1
```

Step 2

This step modifies SAP variant **SBT2**.

The following figure illustrates the variant definition file for variant **SBT2**.

```
/* Variant Header statement. */
VARIANT_NAME   = "SBT2"
   REPORT      = "RSUSR002";

/* User */
SELNAME        = "USER"
   KIND        = "S"
   SIGN        = "I"
   OPTION      = "CP"
   LOW         = " STONEBRANCH1"
   HIGH        = "";
```

The following figure illustrates the command line to modify variant **SBT2**.

```
Usap -dest CF5 -client 800 -userid sapuid -pwd sappwd
     -modify vardefFile2
```

Step 3

This step submits, starts, and monitors a new job - using variant substitution.

> **Note**
> The pre-defined job must have ABAP program **RSUSR002** defined in Step One and Step Two.

The following figure illustrates the variant definition file for variant **SBT2**.

```
usap -dest CF5 -client 800 -userid sapuid -pwd sappwd
     -run -jobname VARSBST1 -jobid 12345678
     -target_variant 1,SBT1;2,SBT2
```

Command Line Options

Command line options used in this example are:

Command Options	Description
-dest	Named set of connection parameters (destination) 'CF5'. These connection parameters are used for communications with the SAP system. The "destinations" are stored in file **saprfc.ini**, which must be in the current directory, or its full path must be specified in environment variable **RFC_INI**.
-client	SAP client number that the Universal Connector will communicate with.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-modify	Specification that Universal Connector will issue the modify command for the specified definition file.
-run	Specification that Universal Connector will run the new job.
-jobname	Job name of the SAP job that will be used as a template.
-jobid	Job ID of the SAP job that will be used as a template.
-target_variant	Replacement variants for the ABAP program job steps.

Components

Universal Connector for UNIX

Creating a Variant Definition Using GENERATE VARDEF Command - UNIX

Creating a USAP Variant Definition Using the USAP GENERATE VARDEF Command – UNIX

SAP variants often have many parameters. This can make it tedious and time-consuming to create Universal Connector variant definitions by hand.

Fortunately, Universal Connector offers a function that will generate a complete variant definition based on a pre-existing template variant on the SAP system. The generated variant definition can then be used with the Universal Connector sub or modify command to prepare a variant for a job run.

The following example demonstrates the use of the generate vardef command. It will generate a complete Universal Connector variant definition based on the pre-existing variant **SBT1** of ABAP program **RSBDSUB**. The generated variant definition will contain all the information required to reproduce the original template variant.

> ⚠ **Note**
> This example assumes (and requires) that a variant named **SBT1** exists for ABAP program **RSBDCSUB**.

The following figure illustrates the command used to generate a Universal Connector variant definition.

```
usap -dest CF5 -client 800 -userid sapuid -pwd sappwd -generate vardef
     -abapname RSBDCSUB -variant SBT1
```

Command Line Options

Command line options used in this example are:

Command Options	Description
-dest	Named set of connection parameters (destination) 'CF5'. These connection parameters are used for communications with the SAP system. The "destinations" are stored in file **`saprfc.ini`**, which must be in the current directory, or its full path must be specified in environment variable **RFC_INI**.
-client	SAP client number that the Universal Connector will communicate with.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-generate	Instructs Universal Connector to generate the specified variant definition.
-abapname	Name of the ABAP program that the template variant belongs to.

-variant	Name of the variant that Universal Connector will use as a template for generation.

Components

Universal Connector for UNIX

Creating Job Definition Using GENERATE JOBDEF Command - UNIX

Creating a Universal Connector Job Definition Using the GENERATE JOBDEF Command – UNIX

SAP jobs offer many configuration options. Manually creating Universal Connector job definitions that utilize many configuration options can be tedious and time consuming.

Fortunately, Universal Connector offers a function that will generate a complete job definition based on a pre-existing template job on the SAP system. The generated job definition can then be modified, if needed.

The following example demonstrates the use of the generate jobdef command. It will generate a complete Universal Connector job definition based on the pre-existing job **USP_TEMPLATE_1** with job id **12345678**. The generated job definition will contain all the information required to create a new SAP job definition equivalent to the template job.

> **Note**
> This job assumes (and requires) that a job already exists on the SAP system with:
>
> - Job Name: **USP_TEMPLATE_1**
> - Job ID: **12345678**

The following figure illustrates the command used to generate a Universal Connector job definition.

```
usap -dest CF5 -client 800 -userid sapuid -pwd sappwd -generate jobdef
     -jobname USP_TEMPLATE_1 -jobid 12345678
```

Command Line Options

Command line options used in this example are:

Command Options	Description
-dest	Named set of connection parameters (destination) 'CF5'. These connection parameters are used for communications with the SAP system. The "destinations" are stored in file **`saprfc.ini`**, which must be in the current directory, or its full path must be specified in environment variable **RFC_INI**.
-client	SAP client number that the USAP will communicate with.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-generate	Instructs Universal Connector to generate the specified variant definition.
-jobname	Name of the SAP job that will be used as a template for generation.

-jobid	Job ID of the SAP job that will be used as a template for generation.

Components

Universal Connector for UNIX

Indesca - Web Services Execution

Web Services Execution - Overview

Overview

The Web Services Execution feature of Indesca enables you to extend its remote execution functionality to Internet and message-based workload and create file-based events from inbound Internet and message-based application messages.

Outbound Implementation

The outbound implementation of Indesca’s web services execution – Universal Command Agent for SOA – provides the ability to extend Indesca’s workload execution and management features to Internet and message-based workload.

The Internet and message-based protocols are supported by the HTTP Connector, the SOAP Connector, the JMS Connector, and the MQ Connector. In addition, you can execute or batch workload in the WebSphere XD environment using the XD Connector.

Universal Command Agent for SOA gets 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.

It can be initiated from a variety of sources, regardless of platform, such as one or more job scheduling systems, workflow engines, or EAI tools, as well as from business applications and end users.

Indesca enables you to:

1. Consolidate your Internet and message-based workload within your current Enterprise Scheduling environment.
2. Use your existing scheduler, or other workload management applications, along with your new or existing Workload Automation components.
3. Use your existing development, test, and production business processes.
4. Use a single point of workload execution that is not tied to specific vendor hardware or software platforms.

(See Web Services Execution - Examples.)

Inbound Implementation

The inbound implementation of Indesca’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 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.

(See Web Services Execution - Examples.)

Web Services Execution - Examples

Web Services Outbound Examples - Windows and UNIX

- Using Indesca to Publish to a SOA Workload - Windows and UNIX
- Message Payload for SOAP - Windows and UNIX
- Logging Configuration - Windows and UNIX
- UAC HTTP Form - Windows and UNIX

Web Services Outbound Examples - z/OS

- Outbound SOAP Implementation - z/OS

Web Services Inbound Examples - Windows and UNIX

- Inbound JMS Implementation - Windows and UNIX
- Inbound SOAP Implementation - Windows and UNIX

Using Indesca to Publish to a SOA Workload - Windows and UNIX

Basic Structure of Using Indesca to Publish to a SOA Workload

The following figure illustrates the basic structure of using Indesca to publish to a SOA workload.

```
ucmd -script options.txt -script_type SERVICE -host [hostname or IP Address]  -userid username -pwd
password -stdin
     -localfile payload_file.txt
```

Command Line Options

The command line options used in this example are:

Option	Description
-script	File containing the options that instruct the container what type of workload publish
-script_type	Type of script specified by -script.
-host	hostname or IP Address of the UAC Container.
-userid	Valid username.
-pwd	Valid password for userid.
-stdin	Start of stdin options.
-localfile	Redirect the standard file from or to <filename>

The contents of the file **`options.txt`** define the type of workload being published to (see Example Workloads, below).

Components

Universal Command

Universal Command Agent for SOA

Example Workloads

Below are examples of various workloads.

JMS ActiveMQ Workload

```
-jmsdestination dynamicQueues/Soatest2TestQueue1
-jmsconnectionfactoryname ConnectionFactory
-protocol JMS
-serviceurl tcp://soatest2:61616
-timeoutsec 120
-jmscontextfactoryname org.apache.activemq.jndi.ActiveMQInitialContextFactory
-mep Publish
```

JMS Websphere Workload

```
-jmsdestination jms/Soatest2TestQueue1
-jmsconnectionfactoryname jms/SBSConnectionFactory
-protocol JMS
-serviceurl iiop://soatest2:2809
-timeoutsec 120
-jmscontextfactoryname com.ibm.websphere.naming.WsnInitialContextFactory
-jmspropertiesfile websphere_only.properties.xml
-mep Publish
```

Note
The preceding example utilizes a properties file that is located on the UAC server. The following illustrates the contents of the listed properties file:

```
<?xml version="1.0" encoding="UTF-8"?>
<sb:JMSProperties xmlns:sb="http://com.stonebranch/UAI/JMSProperties"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://com.stonebranch/UAC/JMSProperties
JMSProperties.xsd ">
                        <sb:Property>
<sb:Name>jms.initialcontext.com.ibm.CORBA.ORBInit</sb:Name>
                                <sb:Value>com.ibm.ws.sib.client.ORB</sb:Value>
                        </sb:Property>
</sb:JMSProperties>
```

XD Workload

```
-xdcmd SUBMIT
-xdcmdid XDJOB1
-servicepassword xdservicepass
-protocol XDSOAP
-serviceurl http://soatest2:9080/LongRunningJobSchedulerWebSvcRouter/services/JobScheduler
-serviceusername xdusername
-timeoutsec 120
-mep REQUEST
```

MQ Series Workload

```
-mqqueuemanagername MyQueueManager
-mqqueuename UpsQaQueue
-mqhost soatest2
-mqchannel UpsQaChannel
-protocol mq
-timeoutsec 120
-mep publish
```

Message Payload for SOAP - Windows and UNIX

Message Payload for SOAP

The following figure illustrates an example of a basic message payload for SOAP.

```
<tns:ValidateZip xmlns:tns="http://webservicemart.com/ws/">
     <tns:ZipCode>30004</tns:ZipCode>
</tns:ValidateZip>
```

The first line contains:

- Name of the operation (in this case, **ValidateZip**)
- Location of the web service providing the operation (in this case, http://webservicemart.com/ws/).

The second line contains:

- Tag for the value **ZipCode**.
- Actual value, **30004**, that the web service needs to operate.

The third line is the closing tag for the operation named in the first line (in this case, **ValidateZip**).

The other items, such as **tns** and **xmlns**, are namespace identifiers. In most cases, the application developers will provide you with the message payload.

SOAP Response

The following figure illustrates the SOAP response that the ValidateZip operation returns.

```
<string>
<result code="200"><item zip="30004" state="GA" latitude="34.11917"
  longitude="-84.30292"/></result>
<string>
```

The first line indicates the type of data being returned (in this case, string data).

The second line contains the response from the ValidateZip web service operation. It includes:

- result - root element and indicates the start of the response data
- code - success or error code from the HTTP transaction. A value of **"200"** indicates success.
- item - Element that defines the attributes returned in response to the **ZipCode** value submitted.
- zip - ZIP code that was submitted as part of the request.
- state - State in which the ZIP code is located.
- latitude - Latitude of the ZIP code submitted.
- longitude - Longitude of the ZIP code submitted.

The third line is the closing tag for the response message.

Components

Universal Command Agent for SOA

Logging Configuration - Windows and UNIX

Logging Configuration

The following examples illustrate how to check the logs for information regarding the operation of Universal Command Agent for SOA.

Configuration of the logging operations is done via the following files:

- **uac_log4jConfiguration.xml** file for Universal Application Container (UAC).
- **uai_log4jConfiguration.xml** file for Universal Application Interface (UAI).

The logging levels supported by the logging implementation are:

- TRACE
- DEBUG
- INFO
- WARN
- ERROR (default)
- FATAL

⚠ **Note**
The logging level should be changed only at the request of Stonebranch, Inc. Customer Support, as it can have a huge impact on performance.

Components

Universal Command Agent for SOA

uac_log4jConfiguration.xml Example

⚠ **Note**
Lines starting with **<!--** begins a commented string. These comments end with **-->**.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/" debug="false" threshold="all">
  <appender name="RollingFileAppender"
  class="org.apache.log4j.RollingFileAppender">
    <param name="File" value="/var/opt/universal/log/uac/uac.log"/>
    <param name="MaxFileSize" value="1000KB"/>
    <param name="MaxBackupIndex" value="4"/>
    <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="%d %-5p %-17c{2} (%30F:%L) %3x -
    %m%n"/>
    </layout>
  </appender>
  <appender name="LF5Appender" class="org.apache.log4j.lf5.LF5Appender">
    <param name="MaxNumberOfRecords" value="1000"/>
  </appender>
  <appender name="NTEventLogAppender" class="org.apache.log4j.nt.NTEventLogAppender">
    <param name="Source" value="UAC"/>
    <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="%c{1} %M - %m%n"/>
    </layout>
  </appender>
  <appender name="ConsoleAppender" class="org.apache.log4j.ConsoleAppender">
    <param name="Target" value="System.err"/>
    <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="%d %-5p %-17c{2} (%30F:%L) %3x -
    %m%n"/>
    </layout>
  </appender>
  <logger name="com.stonebranch" additivity="true">
    <level value="error"/>
  </logger>
  <root>
    <priority value="error"/>
    <!--<appender-ref ref="LF5Appender"/>-->
    <appender-ref ref="RollingFileAppender"/>
    <!--<appender-ref ref="ConsoleAppender"/>-->
    <!--appender-ref ref="NTEventLogAppender"/--->
  </root>
</log4j:configuration>
```

uai_log4jConfiguration.xml Example

Note
Lines starting with **<!--** begins a commented string. These comments end with **-->**.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/" threshold="null" debug="null">
  <appender name="RollingFileAppender"
  class="org.apache.log4j.RollingFileAppender">
    <param name="File" value="/var/opt/universal/log/uai/uai.log"/>
    <param name="MaxFileSize" value="1000KB"/>
    <param name="MaxBackupIndex" value="4"/>
    <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="%d %-5p %-17c{2} (%30F:%L) %3x -
    %m%n"/>
   </layout>
  </appender>
  <appender name="LF5Appender" class="org.apache.log4j.lf5.LF5Appender">
    <param name="MaxNumberOfRecords" value="1000"/>
  </appender>
  <appender name="ConsoleAppender" class="org.apache.log4j.ConsoleAppender">
    <param name="Target" value="System.err"/>
    <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="%d %-5p %-17c{2} (%30F:%L) %3x -
    %m%n"/>
    </layout>
  </appender>
  <logger name="com.stonebranch" additivity="true">
    <level value="error"/>
  </logger>
  <root>
    <priority value="error"/>
    <!--appender-ref ref="LF5Appender" / -->
    <!--<appender-ref ref="RollingFileAppender"/>-->
    <appender-ref ref="ConsoleAppender"/>
  </root>
</log4j:configuration>
```

Outbound SOAP Implementation - zOS

Outbound SOAP Implementation

Outbound SOAP requests are made via a submitted batch job. The batch job utilizes Universal Command to initiate Universal Command Agent for SOA on a Linux server.

The SOAP method used is request / acknowledge, which means that the batch job completes once it has received an acknowledgement from the application that the delivered SOAP message has been received. At this point, Indesca is not aware of the status of any application processes initiated by the delivered SOAP message.

The outbound SOAP message delivered to the application contains the following three parameters:

1. Run Date: Current date in the format YYYY-MM-DD.
2. Request Identifier: Provided by the application.
3. Run Type: Currently, **START** is the only valid value.

JCL

The following JCL will initiate the outbound SOAP request.

```
//TZE025R2 JOB (TEST,CC0KG1500000),'WINDOWS',                           JOB08030
//             CLASS=S,
//             MSGCLASS=R
//*
// JCLLIB ORDER=TEST.SYS5.UNV.SUNVSAMP
//* ********************************************************************
//* * Sample SOA Communication for R1
//* * ******************************************************************
//* * STEPS - FUNCTION
//* * -----   -------------------------------------------------------
//* * SYSIN - Target destination for process / LINUX
//* * INPUT - Universal Command Options to execute SOAP
//* * UNVIN - PAYLOAD being passed to server
//* ********************************************************************
//STEP1    EXEC UCMDPRC
//LOGIN    DD  DISP=SHR,DSN=ZE025.PROD.INDESCA(IDNPSWD)
//SYSIN    DD  DISP=SHR,DSN=ABC.CONTROL.UPARMLIB(HOSTPARM)
//INPUT    DD  DISP=SHR,DSN=ABC.CONTROL.UPARMLIB(SOAPCALL)
//UNVIN    DD  DISP=SHR,DSN=ABC.CC030210.PMS002.STGXML.START
```

This JCL executes the Universal Command JCL procedure.

DD Statements

The DD Statements contain the following:

LOGIN DD

Encrypted password for the Linux Server running Universal Command Agent for SOA. The encrypted file is created with the Universal Encrypt utility.

SYSIN DD

Universal Command runtime parameters:

- -host
  DNS name or IP address of the Linux Server running Universal Command Agent for SOA.
- -encryptedfile
  Specified the DD name that will contain the encrypted password file.
- -script
  Specifies the DD name that will contain the Universal Command Agent for SOA runtime parameters that are passed to the Universal

Command for Agent SOA.
- -script_type
  The value **SERVICE** tells Universal Command that this is a SOA process.

Outbound SOAP Request - SYSIN DD Contents

```
-host                   deveis01
-encryptedfile          LOGIN
-script                 INPUT
-script_type            SERVICE
```

INPUT DD

Universal Command Agent for SOA runtime parameters:

- -protocol
  Indicates which of the supported SOA protocols to use for this request.
- -mep
  The value **REQUEST** tells the Universal Command Agent for SOA that this request is synchronous (two-way and blocked until a reply is sent by the target workload).
- -serviceurl
  Specifies the URL address (internet, network, or file-based) of the target workload.
- -serviceusername
  Specifies the user name to be passed to the target workload for authentication.
- -servicepassword
  Specifies the password to be passed to the target workload for authentication.
- -timeoutsec
  Specifies the length of time - in seconds - to wait for the request to complete.

Outbound SOAP Request - SYSIN DD Contents

```
-protocol SOAP
-mep request
-serviceurl http://asmws2/rbs_ws/services/BatchCtrlSvcWS
-serviceusername dummy
-servicepassword dummy
-timeoutsec 120
```

UNVIN DD

Universal Command for SOA payload. Contains the values for Run Date, Request Identifier and Request Type.

Outbound SOAP Request - SYSIN DD Contents

```
<est:processBatchCtrlSvcTxn
 xmlns:est="http://abcinsurance.com//services/establish-task-facade/">
            <batchctrlsvcReq>
               <ReqHeader>
                  <ReqId>AUT4510021710113870000200</ReqId>
                  <CmdType>request</CmdType>
                  <CmdMode>alwaysRespond</CmdMode>
                  <UserId></UserId>
                  <Passwd></Passwd>
               </ReqHeader>
               <BatchCtrlSvc_ReqRecord>
                  <Action>START</Action>
                  <EODDt>2010-02-17</EODDt>
               </BatchCtrlSvc_ReqRecord>
            </batchctrlsvcReq>
         </est:processBatchCtrlSvcTxn>
```

Components

Universal Command

Universal Command Agent for SOA

Universal Encrypt

UAC HTTP Form - Windows and UNIX

Example of Universal Command HTTP POST with Form Data

The following is an example of Universal Command Manager executing a Universal Command Agent for SOA HTTP POST request with form data. The service request is specified in the script file, `options.txt`, and the HTTP form data is provided as standard input file `form-data.xml`.

```
ucmd -script options.txt -script_type service -host dallas -userid username -pwd password <
form-data.xml
```

Universal Command Options

The Universal Command Manager command line options used in this example are:

Option	Description
-script	File containing the options that define the service request to be executed.
-script_type	Type of script specified by -script, which is a service request in this example.
-host	Host name or IP Address of the Universal Broker where a Universal Application Container (UAC) is executing. The service request is ultimately executed by the UAC component on behalf of the Universal Command Manager.
-userid	Valid user name on the host specified by the -host option.
-pwd	Valid password for the user name specified by the -userid option.

Components

Universal Command 5.1.0 Reference Guide

Universal Command Agent for SOA

Service Options

The service request is specified with Universal Command Agent for SOA options in the Universal Command script file referred to by the -script option. The example above specifies script file `options.txt` on the -script option. The contents of `options.txt` specifies the following HTTP POST request to be executed.

```
-protocol http
-httpmethod post
-httpformdata true
-mep request
-serviceurl http://www.acme.com/cgi-bin/comment-form.cgi
-timeoutsec 60
```

Each of the Universal Command Agent for SOA options are described below.

Option	Description
-protocol	Protocol used for the service request.
-httpmethod	HTTP method, such as GET or POST.
-httpformdata	Specification for whether or not form data is included in the service request.
-mep	Message exchange pattern. HTTP POST always use a Request pattern.
-serviceurl	URL identifying the location of the HTTP request.
-timeoutsec	Number of seconds to wait for a response from the HTTP server.

Form Data

The example HTTP POST request provides the form data as standard input file `form-data.xml`. The form data is formatted as an XML document. The following `form-data.xml` file is an example that provides two name-value pairs that will be part of the HTTP POST request:

```
<?xml version="1.0" encoding="UTF-8"?>
<p:HTTPFormData xmlns:p="http://com.stonebranch/UAI/HTTPFormData"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://com.stonebranch/UAI/HTTPFormData >HTTPFormData.xsd ">
  <p:Property>
    <p:Name>Comments</p:Name>
    <p:Value>You only live once, but if you work it right, once is enough.</p:Value>
  </p:Property>
  <p:Property>
    <p:Name>box</p:Name>
    <p:Value>yes</p:Value>
  </p:Property>
</p:HTTPFormData>
```

See the HTTP_FORM_DATA option for a description of the XML Schema Definition (XSD) for providing form data key-value pairs.

Inbound SOAP Implementation - Windows and UNIX

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:Enve
```

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

Inbound JMS Implementation - Windows and UNIX

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>
```

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>iiop://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.

```
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 "move $(origname).xml $(origname).found"
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.

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

Indesca - Event Monitoring and File Triggering

Event Monitoring and File Triggering - Overview

Overview

The Event Monitoring and File Triggering feature of Indesca 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 examples illustrated in these pages are specific to the operating systems supported by Workload Automation 5 for the Event Monitoring and File Triggering feature of Indesca.

Links to detailed technical information on appropriate Indesca 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 USER_ID (-userid) and USER_PASSWORD (-pwd) configuration options represent the user ID and password of a valid user account defined on **uemhost**.

Event Monitoring and File Triggering - Universal Event Monitor

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 (**uemsrv**), which uses that information to look for an occurrence of the event and test for its completion.

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

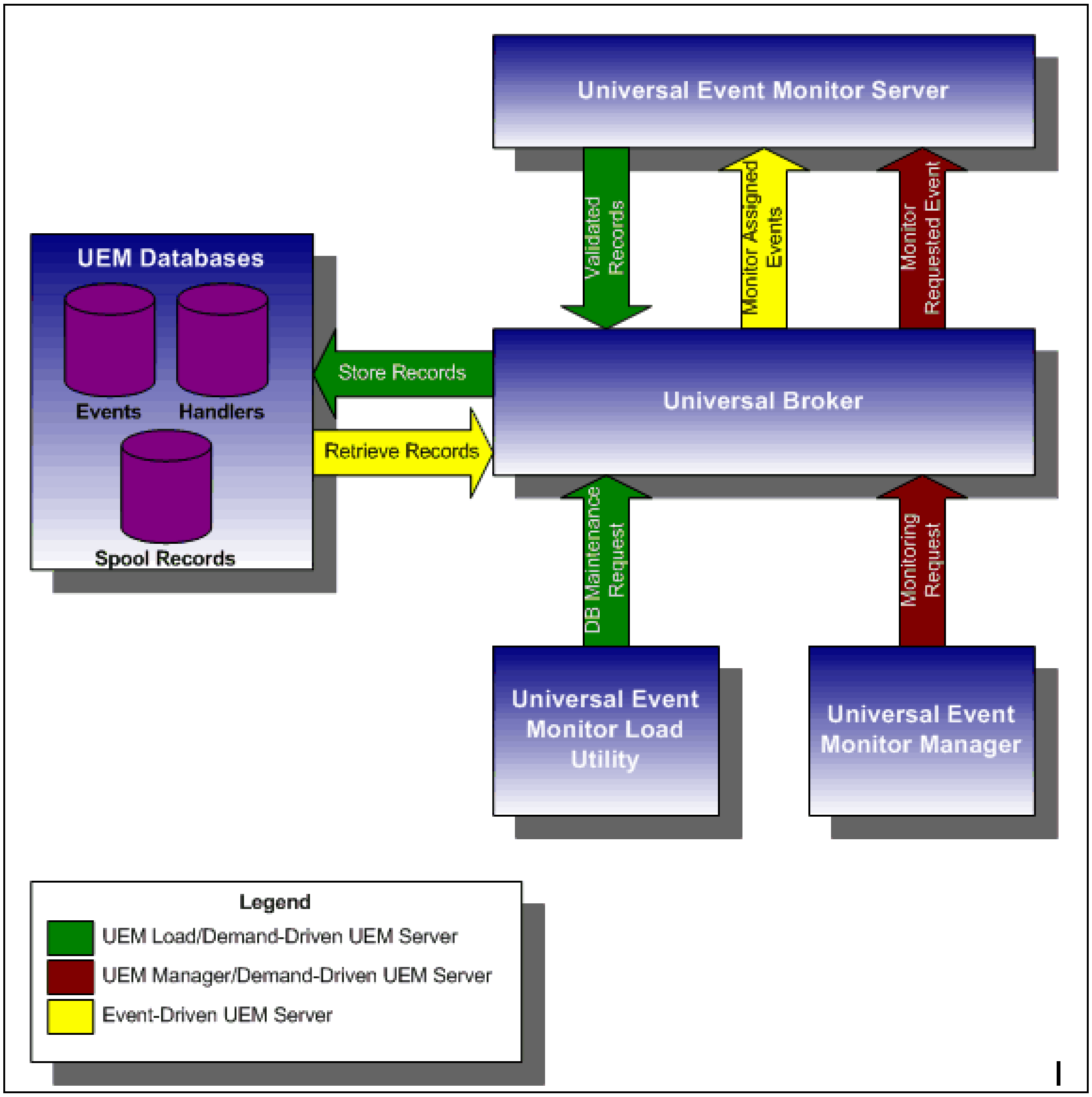
A UEM Server may monitor several local system events simultaneously using records stored in its event definition database. An event-driven UEM Server executes in this manner. An event-driven UEM Server does not require a UEM Manager to initiate a monitoring request, and you may configure it to start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

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

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

High-Level Interaction of UEM Components

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


Universal Event Monitor Server
Validated Records
Monitor Assigned Events
Monitor Requested Event
UEM Databases
Events
Handlers
Spool Records
Store Records
Retrieve Records
Universal Broker
DB Maintenance Request
Monitoring Request
Universal Event Monitor Load Utility
Universal Event Monitor Manager
Legend
🟩 UEM Load/Demand-Driven UEM Server
🟥 UEM Manager/Demand-Driven UEM Server
🟨 Event-Driven UEM Server

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.

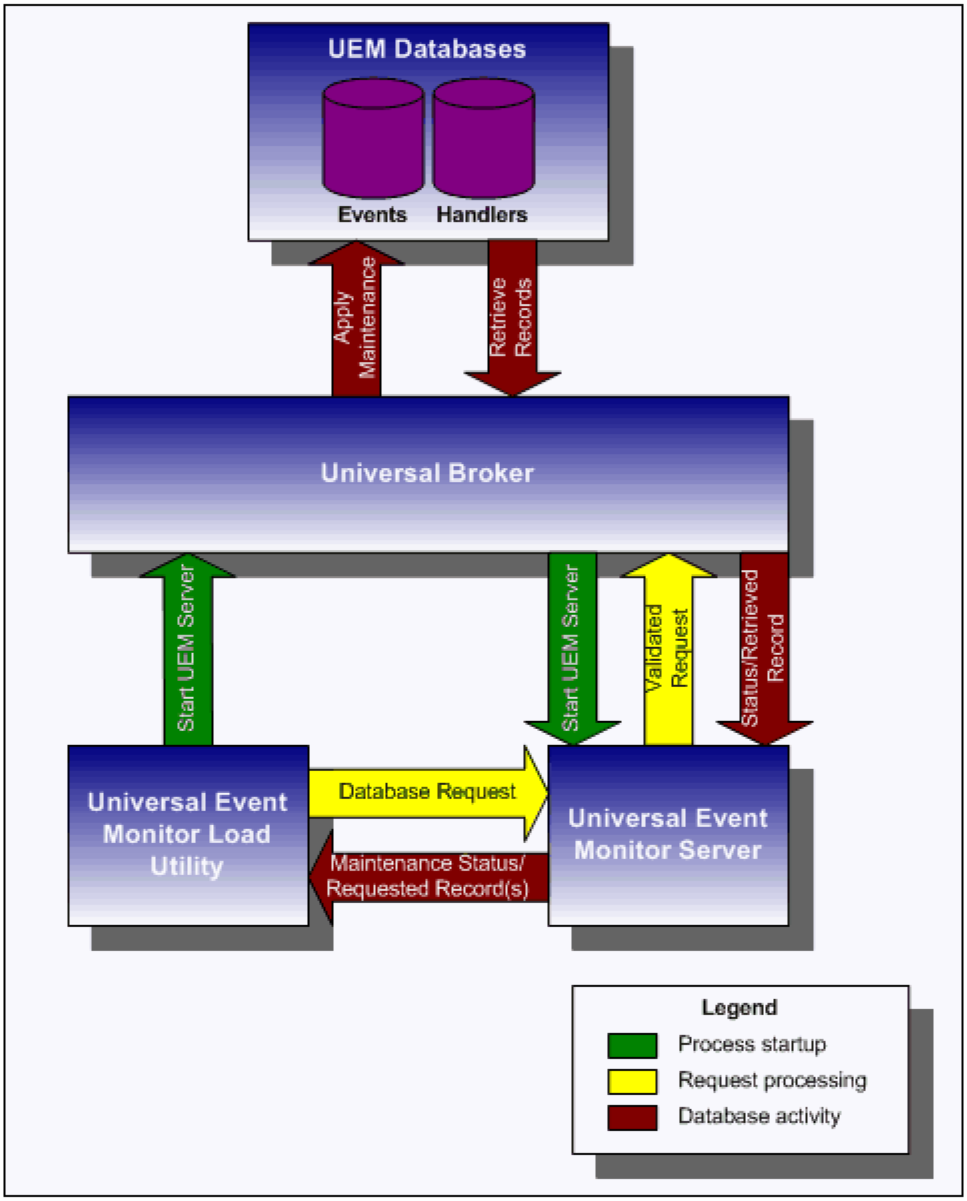
UEMLoad forwards the database request to the UEM Server, which validates it and supplies default values for any required parameters (based upon the type of request) that were not specified from the UEMLoad command line. When a set of complete, valid parameters is available, the UEM Server sends a request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



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:

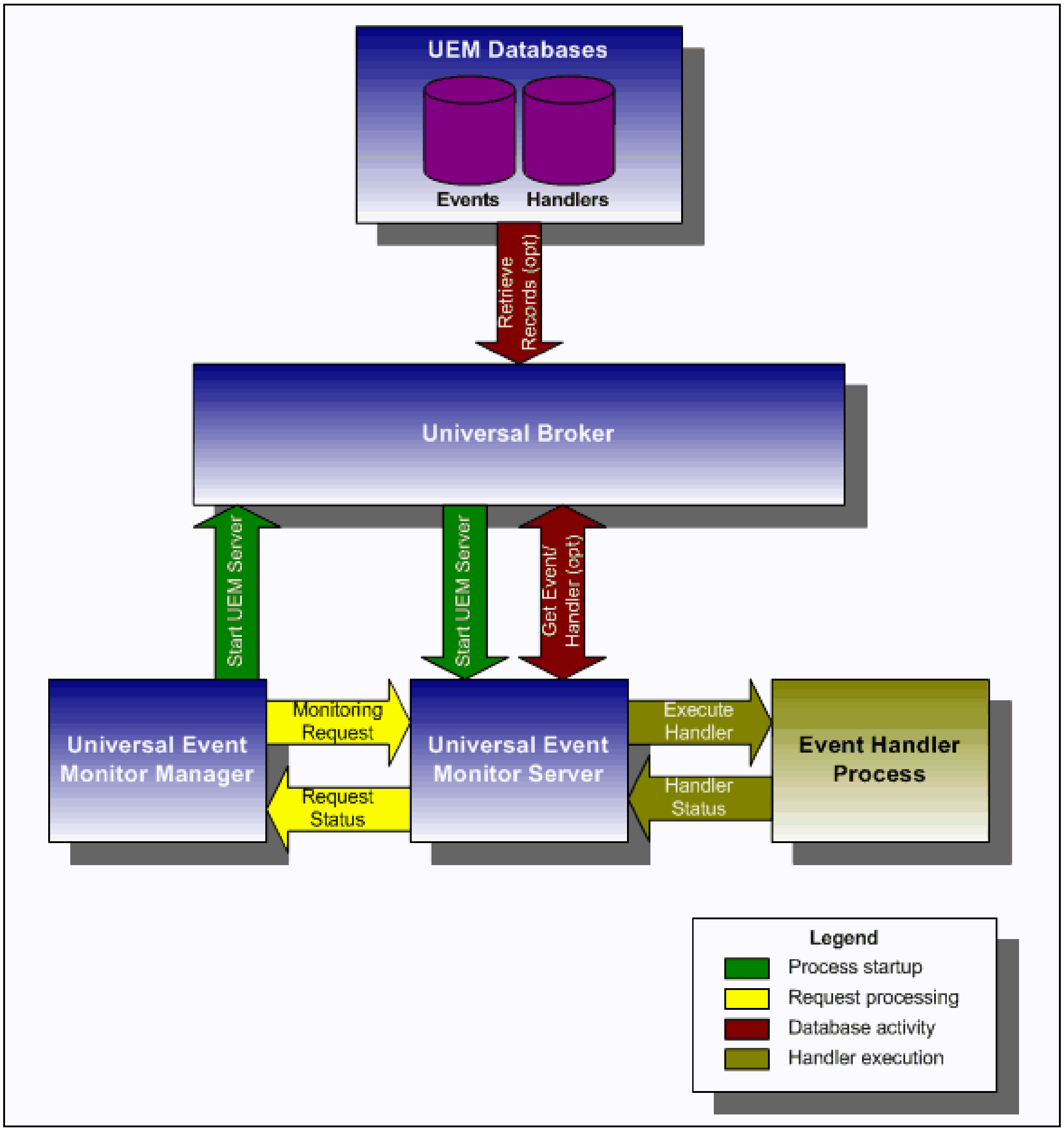
- Required number of expected event occurrences has been detected
- Inactive date and time specified for the event definition elapses.

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.



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.

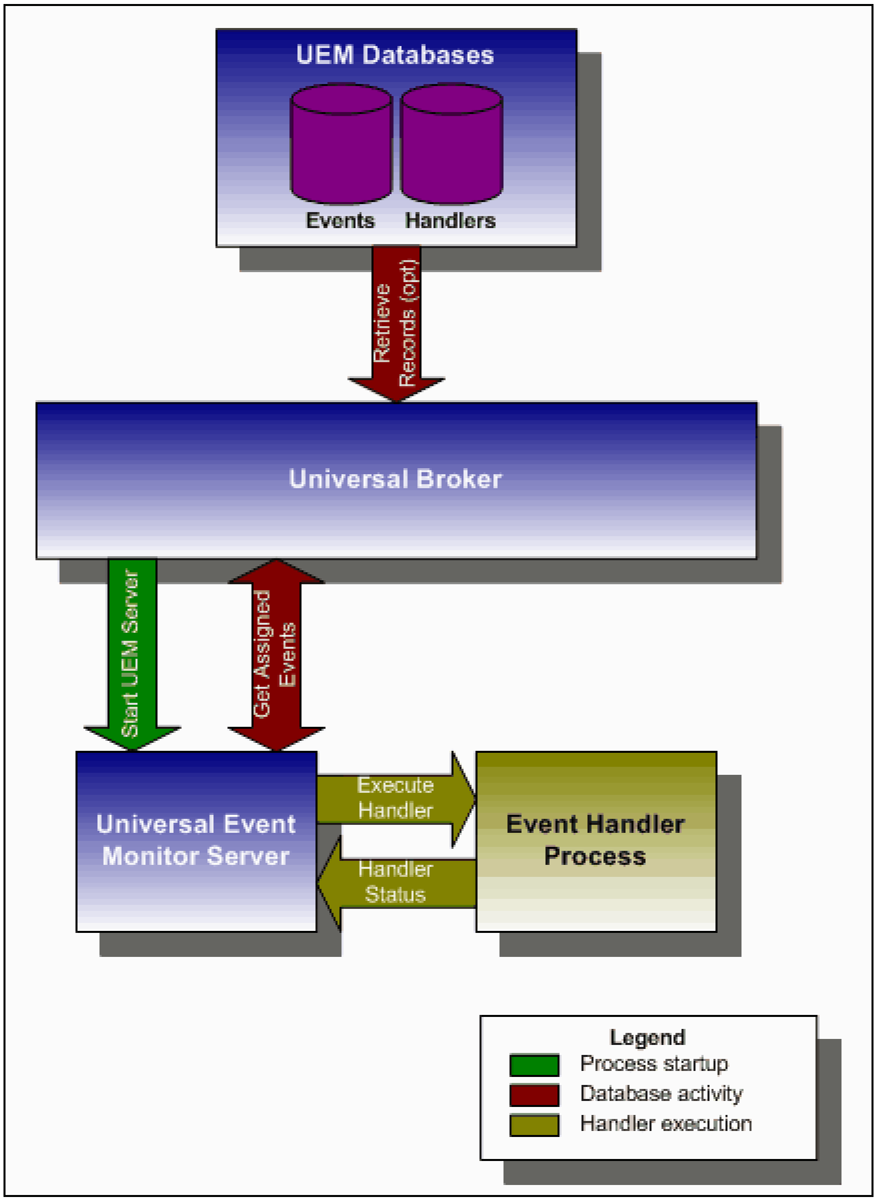
Unless it is stopped manually (using Universal Control), the event-driven UEM Server will continue to run as long as the Broker remains active. When the Broker stops, it will send a stop request to the UEM Server, instructing it to shut itself down.

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

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 - Overview.

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 - zOS

- Starting an Event-Driven UEM Server - zOS
- Refreshing an Event-Driven UEM Server - zOS
- 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 z/OS) - 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 z/OS) - 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
-event_type	Type of event to monitor.
-filespec	Name or pattern of the file whose creation should be detected and tracked for completion.
-wait	Forces the UEM Manager to wait for the completion of the UEM Server.
-inact_date_time	Date and time at which the state of the monitored event should be made inactive.
-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.

-host	List of one or more hosts upon which a command may run.
-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.
-max_count	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 myscript
/*
```

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-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.
-handler_opts	Forces the UEM Manager to wait for the completion of the UEM Server.
-host	List of one or more hosts upon which a command may run.
-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.
-script	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
-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.
-expired	Event state that, when encountered, will result in the execution of the associated event handler.
-cmd	Complete path to an application file or remote script that should be executed on behalf of the event handler.

-host	List of one or more hosts upon which a command may run.
-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.

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

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
-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.

-script	Complete path to a local script file that contains one or more system commands that should be executed on behalf of the event handler.
-handler_opts	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
-host	List of one or more hosts upon which a command may run.
-event_type	Type of event to monitor.
-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.
-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.
-expired	Event state that, when encountered, will result in the execution of the associated event handler.
-cmd	Complete path to an application file or remote script that should be executed on behalf of the event handler.
-options	Values that are passed as command line arguments to a particular handler specified for a given EVENT_STATE.

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
-add	Writes one or more new event definition and/or event handler records to the appropriate database.
-event_id	Identifier that uniquely identifies an event definition record.
-event_type	Type of system event represented by the event definition record.
-filespec	Name of a file to monitor.
-triggered_id	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
-add	Writes one or more new event definition and/or event handler records to the appropriate database.
-handler_id	Identifier that uniquely identifies an event handler record.
-encryptedfile	Complete path to a file encrypted with Universal Encrypt.
-cmd	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
-export	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.
-deffile	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
-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.

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 rmthost –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
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.

inact_date_time	Date and time at which UEM will stop monitoring an event definition.
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.
rename_file	Specification for whether or not UEM should rename a monitored file when an event occurrence is triggered.
rename_filespec	Specification for how a file should be renamed when an event occurrence is triggered.
handler_id	Identifier that uniquely identifies an event handler record.
handler_type	Type of process executed for the event handler, based on the contents of the USER_COMMAND and USER_SCRIPT parameters.
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.
script_type	Type of script statements contained in the action field of the event handler record.
cmd	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

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
-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.

-script	Complete path to a local script file that contains one or more system commands that should be executed on behalf of the event handler.
-handler_opts	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 "/uem files"'** command to a file in uemuser's home directory called **uemtest.log**.

```
uem -host uemhost -event_type file -filespec uemtest.dat
-userid uemuser -pwd uemusers_password
-inact_date_time +1
-expired -cmd 'ls -alR “/uem files”' -options '>uemtest.log 2>&1'
```

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.
-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.
-inact_date_time	Date and time at which the state of the monitored event should be made inactive.
-expired	Event state that, when encountered, will result in the execution of the associated event handler.

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

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
-add	Writes one or more new event definition and/or event handler records to the appropriate database.
-event_id	Identifier that uniquely identifies an event definition record.
-event_type	Type of system event represented by the event definition record.
-filespec	Name of a file to monitor.
-triggered_id	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
-add	Writes one or more new event definition and/or event handler records to the appropriate database.
-handler_id	Identifier that uniquely identifies an event handler record.
-encryptedfile	Complete path to a file encrypted with Universal Encrypt.
-cmd	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
-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 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
-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

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
-export	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.
-deffile	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
-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.

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 rmthost –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.

inact_date_time	Date and time at which UEM will stop monitoring an event definition.
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.
rename_file	Specification for whether or not UEM should rename a monitored file when an event occurrence is triggered.
rename_filespec	Specification for how a file should be renamed when an event occurrence is triggered.
handler_id	Identifier that uniquely identifies an event handler record.
handler_type	Type of process executed for the event handler, based on the contents of the USER_COMMAND and USER_SCRIPT parameters.
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.
script_type	Type of script statements contained in the action field of the event handler record.
cmd	Command to execute on behalf of the event handler.

Components

UEMLoad Utility for UNIX

Indesca - Encryption

Encryption - Overview

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 Stonebranch 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 Workload Automation 5, 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 Indesca 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

Encryption Examples - HP NonStop

- Creating Encrypted Command File for HP NonStop

Creating Encrypted Command File for zOS

Creating Encrypted Command File for z/OS

Command File

In this example, a Universal Command command file named **MY.CLEAR.CMDFILE** 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.CMDFILE
//UNVOUT   DD  DISP=SHR,MY.ENCRYPT.CMDFILE
//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
-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.
-encryptedfile	Encrypted command file.
-script	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

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
-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 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
-key	Encryption key used by the encryption algorithm.
-aes	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
-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.
-encryptedfile	Encrypted command file.
-cmd	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

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
-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 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
-key	Encryption key used by the encryption algorithm.
-aes	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
-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.
-encryptedfile	Encrypted command file.
-cmd	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

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

Creating Encrypted Command File for HP NonStop

Creating Encrypted Command File for HP NonStop

Command File

In this example, a command file named **`cmdfile`** 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.

Encryption Command

The following command encrypts the command file using an encryption key **MYKEY123**:

```
run uencrypt /IN cmdfile, OUT encfile/ –key MYKEY123
```

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

Command Line Options

The command line options used in this example are:

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

Contents of Encrypted File

The following figure below illustrates the contents of **`encfile`**.

```
# Universal Encrypt
# Created on Mon Jul 14 16:47:50 2011
# Created by uencrypt 2.1.1 Level 0

4F4813F7767318C3B1FB016F95B5FD07A6F90A787D9643A03C36503E761DF84AB64FF8877C76F9
8FDBEA1CE672A2DE943CE81BC1C159ABB01D0EC9E52E04A8C21A0269BE85F8443C1A5543901851
C29BE8223471A6BCD498163CD40D1E1866B4
```

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 IBM i

Universal Encrypt

Indesca - Command References

Overview

A command reference is a file residing on the system Universal Command (UCMD) Server installed that contains a pre-defined command or script to be executed at the request of a Universal Command Manager.

The UCMD Manager requests execution of a command reference by specifying:

- The name of the command reference with the COMMAND option.
- The command type as a command reference with the COMMAND_TYPE option.

The UCMD Server finds the command references in the configured command reference location, as specified in the UCMD Server CMD_REFERENCE_DIRECTORY option. If the command reference is found, the UCMD Server executes the command or script defined in the command reference.

The UCMD Manager does not provide its own command or script to execute. It only provides the name of a command reference. This provides the ability to precisely control what UCMD Managers can execute on the server. The UCMD Manager does not have the ability to specify the commands executed on the system. It can only reference a pre-defined command reference that has been defined by the UCMD administrator for the system.

A command reference also can specify that the command or script accepts options or command line arguments from the UCMD Manager.

Command Reference Sections

A command reference is a sequential file containing two sections:

1. Options Section
2. Command Section

The sections are separate with a logical end-of-file marker. The logical end-of-file market is the string **<eof>**.

Options Section

The Options section contains Command Reference Options that define the command reference and specify operational characteristics. It also can contain comments.

Syntax Rules

Syntax rules for the Options section are:

1	Lines starting with a hash character (#) are comments.
2	Blank lines are ignored.
3	Options are specified with a Name followed by a Value.
4	Names are prefixed with a hyphen (-).
5	Names are case-insensitive.

6	Values are case-insensitive.
7	One or more spaces must separate the Name and Value.
8	Options section is ended with <eof> at the start of a line.

Command Reference Options

The following table provides a description and values for all command reference options.

Name	Description	Values
-format	Format of the Command Section. This option is required.	Valid values are: • **cmd** Command section contains a command. • **script** Command section contains a script. The script is defined in the same format as if it was being defined normally in a self-contained file.
-type	Type of command or script in the Command Section. This option is not required.	Values depend on the UCMD Server operating system. See the UCMD Server COMMAND_TYPE and SCRIPT_TYPE options for description of valid values.
-allow_options	Specification for whether or not the command or script accepts user-provided options or command line arguments from the UCMD Manager. This option is not required.	Valid values are: • **yes** Options are passed. ◦ For commands, the UCMD Manager options are concatenated to the end of the command. ◦ For scripts, the UCMD Manager options are passed to the script as command line arguments. • **no** Options are not passed. This is the default.

Command Section

The Command section contains the command or script (as specified by the **-format** command reference option) to be executed.

Commands

Commands are single commands that are executed based on the **-type** option.

The command may span multiple lines. No line continuation characters are required. All lines will be concatenated together to form one command line.

Scripts

Scripts are read verbatim and placed into a temporary file. The temporary file is passed to the script processor for the appropriate script type, as specified by the **-type** option.

Command Reference Example

The following figure is an example command reference that executes a Windows DIR command. The two sections are separated with a logical end-of-file marker, **<eof>**.

```
# Comments describing the command reference.
#
# Execute the DIR command.
#

-format         cmd
-type           shell
-allow_options  no

<eof>

DIR
```

Command Reference Request

A UCMD Manager requests the execution of a command reference just as it does for other command types: via the COMMAND_TYPE option. A COMMAND_TYPE value of `cmdref` specifies that the COMMAND option value is referring to a command reference.

Examples

The following examples illustrate how a UCMD Manager for UNIX would invoke a command reference.

Executing a command reference with no options

The following example executes a command reference named `cmd100`. The `-cmd` option specifies the name of the command reference file on the remote system and the `-cmd_type` option specifies that `-cmd` is referring to a command reference.

```
ucmd –cmd cmd100 –cmd_type cmdref ...
```

Executing a command reference with options

The following example executes a command reference named `cmd100`. The `-cmd` option specifies the name of the command reference file on the remote system and the `-cmd_type` option specifies that `-cmd` is referring to a command reference. Two command line options are passed to the command reference, `opt1` and `opt2`.

```
ucmd –cmd “cmd100 opt1 opt2” –cmd_type cmdref ...
```

The actual format of the options, `opt1` and `opt2`, depend on the command or script being executed in the command reference. In some cases, the options may be required to be comma separated and, other cases, space separated.

Command Line Options

The command line options used in these examples are:

Option	Description
-cmd	Command to execute on the remote system.
-cmd_type	Type of command specified by -cmd.

UACL Rules

Command references can be protected with Universal Access Control List (UACL) rules. Access can be denied or allowed based on the command reference name (see Universal Access Control List for details).

Indesca - Copying Files to and from Remote Systems

Indesca - Copying Files to and from Remote Systems Overview

Introduction

Indesca provides for the copying of files to and from remote systems via its Universal Copy utility.

The following examples illustrate file copying for all supported platforms.

Copying Files Examples - z/OS

- Copy from Local z/OS to Remote Windows
- Copy from Remote Windows to Local z/OS
- Copy from Local z/OS to Remote UNIX
- Copy from Remote UNIX to Local z/OS
- Copy from Local z/OS to Remote IBM i
- Copy from Remote IBM i to Local z/OS
- Copy from Local z/OS to Remote HP NonStop
- Copy from Remote HP NonStop to Local z/OS
- Third-Party Copy via Local z/OS, from Windows to UNIX
- Third-Party Copy via Local z/OS, from UNIX to Windows
- Third-Party Copy via Local z/OS, from Windows to Windows
- Third-Party Copy via Local z/OS, from UNIX to UNIX
- Copy from Local z/OS to Remote System (in Binary)
- Copy from Remote System to Local zOS (in Binary)
- Copy from Local z/OS to Remote z/OS
- Copy from Remote z/OS to Local z/OS
- Copy from Local z/OS to Remote Windows (with Windows Date Variables)
- Copy from Local z/OS to Remote UNIX (with UNIX Date Variables)
- Copy from Remote UNIX to Local z/OS Using cat Command

Copying Files Examples - Windows

- Copy from Remote UNIX to Local Windows
- Copy From Local Windows to Remote UNIX
- Copy from Remote UNIX to Local Windows Using the UNIX cat Command

Copying Files Examples - UNIX

- Copy from Local UNIX to Remote Windows
- Copy Encrypted File from Local UNIX to Remote Windows
- Copy from Remote Windows to Local UNIX
- Copy Encrypted File from Remote Windows to Local UNIX

Copying Files Examples - IBM i

- Copy from Remote Windows to Local IBM i via UCMD Manager
- Copy from Remote IBM i to Local Windows via UCMD Manager
- Copy from Local Windows to Remote IBM i via UCMD Manager
- Copy from Local IBM i to Remote Windows via UCMD Manager

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

Copying Files Examples - HP NonStop

- Copy from Remote Windows to Local HP NonStop via UCOPY
- Copy from Local HP NonStop to Remote Windows via UCOPY
- Copy from Remote Windows to Local HP NonStop (using STDOUT) - 1
- Copy from Remote Windows to Local HP NonStop (using STDOUT) - 2
- Copy from Local HP NonStop to Remote Windows (using STDIN)
- Copy from Local HP NonStop to Remote Windows (using STDOUT)

Copy from Local zOS to Remote Windows

Copy from Local z/OS to Remote Windows via Universal Copy

The following figure illustrates the copying of a file from a local z/OS system to a remote Windows system.

```
//S1 EXEC UCMDPRC
//UNVIN    DD DISP=SHR,DSN=hlq.input.file
//LOGONDD  DD DISP=SHR,DSN=hlq.userid(userid)
//SCRIPTDD DD *
@echo off
ucopy –mode text –output C:\OUTPUT.FILE
//SYSIN    DD *
-script SCRIPTDD
-encryptedfile LOGONDD
-host dallas
/*
```

The JCL procedure **UCMDPRC** is used to execute the command. The command is sent to a remote system named **dallas** for execution.

The -mode option (value **text**) is used with the **ucopy** command to force end-of-line character interpretation. The -output option is used with the **ucopy** command to direct the standard out to a local data set on the remote server.

Additional command line options are read from the encrypted file allocated to DD **LOGONDD**.

The file is copied as a text file, since the default transfer mode for standard files is text.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	DD from which to read a script file. The script file is sent to the remote system for execution.
-encryptedfile	DD from which to read an encrypted command options file.
-host	Directs the command to a computer with a host name of **dallas**.

Components

Universal Command Manager for z/OS

Universal Copy

Copy from Remote Windows to Local zOS

Copy from Remote Windows to Local z/OS via Universal Copy

The following figure illustrates the copying of a file from a remote Windows system to a local z/OS system.

```
//S1 EXEC UCMDPRC
//UNVOUT   DD DISP=SHR,DSN=hlg.output.file
//LOGONDD  DD DISP=SHR,DSN=hlq.userid(userid)
//SCRIPTDD DD *
@echo off
ucopy –mode text C:\INPUT.FILE
//SYSIN    DD *
-script SCRIPTDD
-encryptedfile LOGONDD
-host dallas
/*
```

The JCL procedure UCMDPRC is used to execute the command. The command is sent to a remote system named **dallas** for execution. The **UNVOUT** DD specifies a local data set to use for the standard output of the remote command. The -mode option (value = text) is used with the **ucopy** command to force end-of-line character interpretation.

Additional command line options are read from the encrypted file allocated to DD **LOGONDD**.

The file is copied as a text file, since the default transfer mode for standard files is text.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	DD from which to read a script file. The script file is sent to the remote system for execution.
-encryptedfile	DD from which to read an encrypted command options file.
-host	Directs the command to a computer with a host name of **dallas**.

Components

Universal Command Manager for z/OS

Universal Copy

Copy from Local zOS to Remote UNIX

Copy from Local z/OS to Remote UNIX via Universal Copy

The following figure illustrates the copying of a file from a local z/OS system to a remote UNIX system.

```
//S1 EXEC UCMDPRC
//UNVIN    DD DISP=SHR,DSN=hlq.input.file
//LOGONDD  DD DISP=SHR,DSN=hlq.userid(userid)
//SCRIPTDD DD *
/opt/universal/bin/ucopy –mode text \
-output /usr/output.file
//SYSIN    DD *
-script SCRIPTDD
-encryptedfile LOGONDD
-host dallas
```

The JCL procedure **UCMDPRC** is used to execute the command. The command is sent to a remote system named **dallas** for execution. The -output option is used with the **ucopy** command to direct the standard out to a local data set on the remote server. The path to the **ucopy** binary must be specified if the directory is not defined in the user's path environmental variable. The -mode option (value **text**) is used with the **ucopy** command to force end-of-line character interpretation.

Additional command line options are read from the encrypted file allocated to DD **LOGONDD**.

The file is copied as a text file, since the default transfer mode for standard files is text.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	DD from which to read a script file. The script file is sent to the remote system for execution.
-encryptedfile	DD from which to read an encrypted command options file.
-host	Directs the command to a computer with a host name of **dallas**.

Components

Universal Command Manager for z/OS

Universal Copy

Copy from Remote UNIX to Local zOS

Copy from Remote UNIX to Local z/OS via Universal Copy

The following figure illustrates the copying of a file from a remote UNIX system to a local z/OS system.

```
//S1 EXEC UCMDPRC
//UNVOUT   DD DISP=SHR,DSN=hlq.output.file
//LOGONDD  DD DISP=SHR,DSN=hlq.userid(userid)
//SCRIPTDD DD *
/opt/universal/bin/ucopy –mode text \
/usr/input.file
//SYSIN    DD *
-script SCRIPTDD
-encryptedfile LOGONDD
-host dallas
/*
```

The JCL procedure **UCMDPRC** is used to execute the command. The command is sent to a remote system named **dallas** for execution. The **UNVOUT** DD specifies a local data set to use for the standard output of the remote command. The -mode option (value = text) is used with the **ucopy** command to force end-of-line character interpretation.

Additional command line options are read from the encrypted file allocated to DD **LOGONDD**.

The file is copied as a text file, since the default transfer mode for standard files is text.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	DD from which to read a script file. The script file is sent to the remote system for execution.
-encryptedfile	DD from which to read an encrypted command options file.
-host	Directs the command to a computer with a host name of **dallas**.

Components

Universal Command Manager for z/OS

Universal Copy

Copy from Local zOS to Remote IBM i

Copy from Local z/OS to Remote IBM i via Universal Copy

The following figure illustrates the copying of a file from a local z/OS system to a remote IBM i system.

```
//S1 EXEC UCMDPRC
//UNVIN    DD DISP=SHR,DSN=hlq.input.file
//LOGONDD  DD DISP=SHR,DSN=hlq.userid(userid)
//SCRIPTDD DD *
STRUCP TOFILE(LIBRARY/OUTPUTFILE)TOMBR(MEMBER)
CPYMODE(*TEXT)
//SYSIN    DD *
-script SCRIPTDD
-encryptedfile LOGONDD
-host dallas
/*
```

The JCL procedure **UCMDPRC** is used to execute the command. The command is sent to a remote system named **dallas** for execution. The TOFILE option is used with the **STRUCP** command to direct the standard out to a local data set on the remote server. The CPYMODE option is used to force end-of-line character interpretation.

Additional command line options are read from the encrypted file allocated to DD **LOGONDD**.

The file is copied as a text file, since the default transfer mode for standard files is text.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	DD from which to read a script file. The script file is sent to the remote system for execution.
-encryptedfile	DD from which to read an encrypted command options file.
-host	Directs the command to a computer with a host name of **dallas**.

Components

Universal Command Manager for z/OS

Universal Copy

Copy from Remote IBM i to Local zOS

Copy from Remote IBM i to Local z/OS via Universal Copy

The following figure illustrates the copying of a file from a remote IBM i system to a local z/OS system.

```
//S1 EXEC UCMDPRC
//UNVOUT   DD DISP=SHR,DSN=hlq.output.file
//LOGONDD  DD DISP=SHR,DSN=hlq.userid(userid)
//SCRIPTDD DD *
STRUCP FRMFILE(LIBRARY/INPUTFILE)FRMMBR(MEMBER)
CPYMODE(*TEXT)
//SYSIN    DD *
-script SCRIPTDD
-encryptedfile LOGONDD
-host dallas
/*
```

The JCL procedure **UCMDPRC** is used to execute the command. The command is sent to a remote system named **dallas** for execution. The **UNVOUT** DD specifies a local data set to use for the standard output of the remote command. The CPYMODE option is used to force end-of-line character interpretation.

Additional command line options are read from the encrypted file allocated to DD **LOGONDD**.

The file is copied as a text file, since the default transfer mode for standard files is text.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	DD from which to read a script file. The script file is sent to the remote system for execution.
-encryptedfile	DD from which to read an encrypted command options file.
-host	Directs the command to a computer with a host name of **dallas**.

Components

Universal Command Manager for z/OS

Universal Copy

Copy from Local zOS to Remote HP NonStop

Copy from Local z/OS to Remote HP NonStop via Universal Copy

The following figure illustrates the copying of a file from a local z/OS system to a remote HP NonStop system.

```
//S1 EXEC UCMDPRC
//UNVIN    DD DISP=SHR,DSN=hlq.input.file
//LOGONDD  DD DISP=SHR,DSN=hlq.userid(userid)
//SCRIPTDD DD *
ucopy –output outputfile –mode text
//SYSIN    DD *
-script SCRIPTDD
-encryptedfile LOGONDD
-host dallas
–server “ -script_type OSS”
/*
```

The JCL procedure **UCMDPRC** is used to execute the command. The command is sent to a remote system named **dallas** for execution. The -output option is used with the **ucopy** command to direct the standard out to a local data set on the remote server. The -mode option (value **text**) is used with the **ucopy** command to generate an EDIT file with a file code of 101. A value of binary (default) will generate a C file with a file code of 180.

Additional options are read from the encrypted file allocated to DD **LOGONDD**.

The file is copied as a text file, since the default transfer mode for standard files is text.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	DD from which to read a script file. The script file is sent to the remote system for execution.
-encryptedfile	DD from which to read an encrypted command options file.
-host	Directs the command to a computer with a host name of **dallas**.
-server	The value **-script_type OSS** is specified to notify the UCMD Server that it is to execute an OSS process, since Universal Copy is a native OSS program.

Components

Universal Command Manager for z/OS

Universal Copy

Copy from Remote HP NonStop to Local zOS

Copy from Remote HP NonStop to Local z/OS via Universal Copy

The following figure illustrates the copying of a file from a remote HP NonStop system to a local z/OS system.

```
//S1 EXEC UCMDPRC
//UNVOUT   DD DISP=SHR,DSN=hlq.output.file
//LOGONDD  DD DISP=SHR,DSN=hlq.userid(userid)
//SCRIPTDD DD *
ucopy –mode text inputfile
//SYSIN    DD *
-script SCRIPTDD
-encryptedfile LOGONDD
-host dallas
–server “ -script_type OSS”
/*
```

The JCL procedure **UCMDPRC** is used to execute the command. The command is sent to a remote system named **dallas** for execution. The **UNVOUT** DD specifies a local data set to use for the standard output of the remote command. The -mode option (value = text) is used with the **ucopy** command to read an EDIT file with a file code of 101. A value of binary (default) will read a C file with a file code of 180.

Additional options are read from the encrypted file allocated to DD **LOGONDD**.

The file is copied as a text file, since the default transfer mode for standard files is text.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	DD from which to read a script file. The script file is sent to the remote system for execution.
-encryptedfile	DD from which to read an encrypted command options file.
-host	Directs the command to a computer with a host name of **dallas**.
-server	The value **-script_type OSS** is specified to notify the UCMD Server that it is to execute an OSS process, since Universal Copy is a native OSS program.

Components

Universal Command Manager for z/OS

Universal Copy

Third-Party Copy via Local zOS, from Windows to UNIX

Third-Party Copy via Local z/OS, from Windows to UNIX via Universal Copy

The following figure illustrates the third-party copying of a file from a local z/OS system, which executes a **ucopy** from Windows to UNIX.

```
//S1 EXEC UCMDPRC
//UNVIN    DD DSN=hlq.userid(#useridunx),DISP=SHR
//LOGONDD  DD DSN=hlq.userid(#useridnt),DISP=SHR
//SCRIPT   DD *
@ECHO ON
:: TRANSFER FROM NT to UNIX
@SET UCOPYPATH=/opt/universal/bin/
@SET OUTPUTFILE=outputfile
@SET INPUTFILE=inputfile
@SET UNIXHOST=unixhost
@SET TEMPUNIXID=c:\temp\tempunixid
@SET MODE=text
ucopy -output %TEMPUNIXID%
ucmd¬
 -cmd " %UCOPYPATH%ucopy –output %OUTPUTFILE%"¬
 < %INPUTFILE% -host %UNIXHOST% -encryptedfile %TEMPUNIXID%¬
 -level info -stdin -mode %MODE%
SET RC=%ERRORLEVEL%
del %TEMPUNIXID%
URC %RC%
//SYSIN    DD *
-script SCRIPT
-encryptedfile LOGONDD
-host NTHOST
-level info
/*
```

All informational messages will be routed to the z/OS manager. The authentication information for the UNIX server must reside on the z/OS.

The file is copied as a text file, since the default transfer mode for standard files is text.

Parameters

The following parameters should be changed to match your information:

Parameter	Description
#USERIDUNX	Encrypted userid and password member for UNIX server
#USERIDNT	Encrypted userid and password member for NT server
UCOPYPATH	Path to UCOPY on the receiving UNIX server
OUTPUTFILE	Path and filename of receiving file on UNIX server

INPUTFILE	Path and file name of sending file on Windows server
UNIXHOST	IP address or hostname of receiving UNIX server
NTHOST	IP address or hostname of sending Windows server
TEMPUNIXID	Temporary file on the Windows server used to house the encrypted logon information for the UNIX server. This file is deleted at the bottom of the script.
MODE	Mode of file transfer (binary/text).

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	DD from which to read a script file. The script file is sent to the remote system for execution.
-encryptedfile	DD from which to read an encrypted command options file.
-host	Directs the command to a computer with a host name of **NTHOST**.
-level	Sets the level of message information.

Components

Universal Command Manager for z/OS

Universal Command Manager for Windows

Universal Command Server for UNIX

Universal Copy

Third-Party Copy via Local zOS, from UNIX to Windows

Third-Party Copy via Local z/OS, from UNIX to Windows via Universal Copy

The following figure illustrates the third-party copying of a file from a local z/OS system, which executes a **ucopy** from UNIX to Windows.

```
//S1 EXEC UCMDPRC
//UNVIN    DD DSN=hlq.userid(#useridnt),DISP=SHR
//LOGONDD  DD DSN=hlq.userid(#useridunx),DISP=SHR
//SCRIPT   DD *
export UCMDPATH=/opt/universal/bin
export UCPYPATH=/opt/universal/bin
export OUTPUTFILE="c:\temp\outputfile"
export INPUTFILE=/tmp/inputfile
export NTHOST=nthostname
export TEMPNTID=/tmp/tempntid
export MODE=text
$UCPYPATH/ucopy -output $TEMPNTID
$UCMDPATH/ucmd \
-cmd "ucopy –output $OUTPUTFILE"< $INPUTFILE \
-host $NTHOST -encryptedfile $TEMPNTID -level info -stdin -mode $MODE
rc=$?
rm $TEMPNTID
exit $rc
//SYSIN    DD *
-script SCRIPT
-encryptedfile LOGONDD
-host unixhostname
-level info
/*
```

All error messages will be routed to the z/OS manager. The authentication information for the NT server must reside on the z/OS.

The file is copied as a text file since the default transfer mode for standard files is text.

Parameters

The following parameters should be changed to match your information:

Parameter	Description
#USERIDUNX	Encrypted userid and password member for UNIX server
#USERIDNT	Encrypted userid and password member for Windows server
UCOPYPATH	Path to UNIX **ucopy** executable
UCMDPATH	Path to UNIX **ucmd** executable
OUTPUTFILE	Path and filename of receiving file

INPUTFILE	Path and file name of sending file
NTHOST	IP address or hostname of receiving Windows server
TEMPNTID	Temporary file on the UNIX server used to house the encrypted logon information for the Windows server.
MODE	Mode of file transfer (binary / text). Default is set to text.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	DD from which to read a script file. The script file is sent to the remote system for execution.
-encryptedfile	DD from which to read an encrypted command options file.
-host	Directs the command to a computer with a host name of **unixhostname**.
-level	Sets the level of message information.

Components

Universal Command Manager for z/OS

Universal Command Manager for UNIX

Universal Command Server for Windows

Universal Copy

Third-Party Copy via Local zOS, from Windows to Windows

Third-Party Copy via Local z/OS, from Windows to Windows via Universal Copy

The following figure illustrates the third-party copying of a file from a local z/OS system, which executes a **ucopy** command from Windows to Windows.

The standard error is read into the UMET utility to verify the existence of the input file. The last step copies standard error to the job log.

```
//S1 EXEC UCMDPRC
//UNVIN    DD DSN=hlq.userid(#nt2logon),DISP=SHR
//LOGONDD  DD DSN=hlq.userid(#ntlogon),DISP=SHR
//UNVERR   DD DSN=hlq.output(stderr),DISP=SHR
//SCRIPT   DD *
@ECHO ON
:: TRANSFER FROM NT to NT
@SET OUTPUTFILE=c:\temp\output.file
@SET INPUTFILE=c:\temp\input.file
@SET NT2HOST=hostname
@SET TEMPNT2ID=c:\temp\userid.enc
@SET MODE=text
ucopy –output %TEMPNT2ID%
ucmd¬
 -cmd "ucopy –output %OUTPUTFILE%" <%INPUTFILE% ¬
 -host %NT2HOST% -encryptedfile %TEMPNT2ID% -level info -stdin -mode %MODE%
SET RC=%ERRORLEVEL%
del %TEMPNT2ID%
URC %RC%
//SYSIN    DD  *
 -script SCRIPT
 -encryptedfile LOGONDD
 -host NTHOST
 -level info   /*
//*
//***********************************************************
//S1 EXEC PGM=UMET,PARM='-TABLE TABLE -LEVEL VERBOSE'
//STEPLIB  DD  DISP=SHR,DSN=hlq.UNV.SUNVLOAD
//SYSPRINT DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
//TABLE    DD  *
  "The system cannot find the file specified." 8
/*
//SYSIN    DD  DISP=SHR,DSN=hlq.output(stderr)
//***********************************************************
//S1 EXEC PGM=IEBGENER
//SYSUT1   DD DISP=SHR,DSN=hlq.output(stderr)
//SYSUT2   DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSIN    DD DUMMY
```

All error messages will be routed to the z/OS manager. The authentication information for the WIndows server must reside on the z/OS.

The file is copied as a text file, since the default transfer mode for standard files is text.

The UMETSTEP step executes the UMET utility. UMET is used to set the condition code field to a value based on message text. The SYSIN DD is the standard error of the first step and the TABLE DD is the table defining which condition code to be used when text is found.

⚠ **Note**
The UMET program is used because native Windows returns a 0 return (exit) code, even when the stdin does not exist. Therefore, the process would end with a 0, even if the input file did not exist. UMET will set the condition code to 8.

The IEBGENER step will copy the standard error file to SYSLOG if the process gets a non-zero condition code.

Parameters

The following parameters should be changed to match your information:

Parameter	Description
#USERIDNT	Encrypted userid and password member for sending Windows server
#USERIDNT2	Encrypted userid and password member for receiving Windows server
OUTPUTFILE	Path and filename of receiving file
INPUTFILE	Path and file name of sending file
NTHOST	IP address or hostname of sending Windows server
NT2HOST	IP address or hostname of receiving Windows server
TEMPNT2ID	Temporary file on the Windows sending server used to house the encrypted logon information for the Windows receiving server.
MODE	Mode of file transfer (binary / text). Default is set to text.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	DD from which to read a script file. The script file is sent to the remote system for execution.
-encryptedfile	DD from which to read an encrypted command options file.
-host	Directs the command to a computer with a host name of **NTHOST**.
-level	Sets the level of message information.

Components

Universal Command Manager for z/OS

Universal Command Manager for Windows

Universal Command Server for Windows

Universal Copy

Third-Party Copy via Local zOS, from UNIX to UNIX

Third-Party Copy via Local z/OS, from UNIX to UNIX via Universal Copy

The following figure illustrates the third-party copying of a file from a local z/OS system, which executes a **ucopy** command from UNIX to UNIX.

```
//S1 EXEC UCMDPRC
//UNVIN    DD DSN=hlq.userid(useridunxr),DISP=SHR
//LOGONDD  DD DSN=hlq.userid(useridunxs),DISP=SHR
//SCRIPT   DD *
export UCOPYPATH=/opt/universal/bin
export UCMDPATH=/opt/universal/bin
export OUTPUTFILE=/outputfile
export INPUTFILE=/inputfile
export UNIXRHOST=receivinghostname
export TEMPUNIXRID=/tmp/unixid.tmp
export MODE=text
$UCOPYPATH/ucopy –output $TEMPUNIXRID
$UCMDPATH/ucmd \
 -cmd "$UCOPYPATH/ucopy –output $OUTPUTFILE" < $INPUTFILE \
 -host $UNIXRHOST -encryptedfile $TEMPUNIXRID -level info -stdin -mode $MODE
rc=$?
rm $TEMPUNIXRID
exit $rc
//SYSIN    DD *
 -script SCRIPT
 -encryptedfile LOGONDD
 -host unixshost
 -level info
/*
```

All error messages will be routed to the z/OS manager. The authentication information for both UNIX servers must reside on the z/OS.

The file is copied as a text file since the default transfer mode for standard files is text.

Parameters

The following parameters should be changed to match your information:

Parameter	Description
UCOPYPATH	Path pointing to the **ucopy** executable on the second UNIX server
UCMDPATH	Path pointing to the **ucmd** executable on the second UNIX server
OUTPUTFILE	Path and filename of receiving file
INPUTFILE	Path and file name of sending file
UNIXSHOST	IP address or hostname of sending UNIX server

UNIXRHOST	IP address or hostname of receiving UNIX server
TEMPUNIXRID	Temporary file on the sending UNIX server used to house the encrypted logon information for the receiving UNIX server.
MODE	Mode of file transfer (binary / text). Default is set to text.
USERUNXR	Points to the userid / password information for the receiving UNIX server.
USERUNXS	Points to the userid / password information for the sending UNIX server.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	Specifies the DD from which to read a script file. The script file is sent to the remote system for execution.
-encryptedfile	Specifies the DD from which to read an encrypted command options file.
-host	Directs the command to a computer with a host name of **unixshost**.
-level	Sets the level of message information.

Components

Universal Command Manager for z/OS

Universal Command Manager for UNIX

Universal Command Server for UNIX

Universal Copy

Copy from Local zOS to Remote System (in Binary)

Copy from Local z/OS to Remote System (in Binary) via Universal Copy

The following figure illustrates the copying of a file from a local z/OS system to a remote system, in binary, with no end-of-line character interpretation.

```
//S1 EXEC UCMDPRC
//UNVIN    DD DISP=SHR,DSN=hlq.input.file
//LOGONDD  DD DISP=SHR,DSN=hlq.userid(userid)
//SCRIPTDD DD *
@echo off
ucopy -output C:\OUTPUT.FILE
//SYSIN    DD *
-script SCRIPTDD
-encryptedfile LOGONDD
-host dallas
-stdin –mode binary
/*
```

The JCL procedure **UCMDPRC** is used to execute the command. The command is sent to a remote system named **dallas** for execution. The -output option used with the **ucopy** command directs the stdout to a local data set on the remote server. The -mode option used with the **ucopy** command defaults to binary, so no end-of-line character interpretation is done. Binary is specified for standard input transfer mode.

Additional options are read from the encrypted file allocated to DD **LOGONDD**.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	DD from which to read a script file. The script file is sent to the remote system for execution.
-encryptedfile	DD from which to read an encrypted command options file.
-host	Directs the command to a computer with a host name of **dallas**.
-stdin	Specification that the options following this one apply to the stdin file.
-mode	Specification for whether transferred data is treated as text or binary.

Components

Universal Command Manager for z/OS

Universal Copy

Copy from Remote System to Local zOS (in Binary)

Copy from Remote System to Local z/OS (in Binary) via Universal Copy

The following figure illustrates the copying of a file from a remote system to a local z/OS system, in binary, with no end-of-line character interpretation.

```
//S1 EXEC UCMDPRC
//UNVOUT   DD DISP=SHR,DSN=hlg.output.file
//LOGONDD  DD DISP=SHR,DSN=hlq.userid(userid)
//SCRIPTDD DD *
@echo off
ucopy C:\INPUT.FILE
//SYSIN    DD *
-script SCRIPTDD
-encryptedfile LOGONDD
-host dallas
-stdout –mode binary
/*
```

The JCL procedure **UCMDPRC** is used to execute the command. The command is sent to a remote system named **dallas** for execution. The **UNVOUT** DD specifies a local data set to use for the standard output of the remote command. The -mode option used with the **ucopy** command defaults to binary, so no end-of-line character interpretation is done. Binary is specified for standard output transfer mode.

Additional options are read from the encrypted file allocated to DD **LOGONDD**.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	DD from which to read a script file. The script file is sent to the remote system for execution.
-encryptedfile	DD from which to read an encrypted command options file.
-host	Directs the command to a computer with a host name of **dallas**.
-stdout	Specification that the options following this one apply to the stdout file.
-mode	Specification for whether transferred data is treated as text or binary.

Components

Universal Command Manager for z/OS

Universal Copy

Copy from Local zOS to Remote zOS

Copy from Local z/OS to Remote z/OS (with Encryption, Compression, and Data Authentication) via Universal Copy

The following figure illustrates the copying of a file from a local z/OS system to a remote z/OS system (with encryption, compression, and data authentication).

```
//STEP1 EXEC UCMDPRC
//UNVIN='DISP=SHR,DSN=MY.PDS(MEMBER)
//LOGONDD  DD DISP=SHR,DSN=MY.LOGON(USERID)
//SCRIPTDD DD *
/opt/universal/bin/ucopy > //'REMOTE.PDS(MEMBER)'
//SYSIN DD *
-script SCRIPTDD
-encryptedfile LOGONDD
-host dallas
-stdin -encrypt yes -compress yes -authenticate yes
/*
```

The JCL procedure **UCMDPRC** is used to execute the command. The command is sent to a remote system named **dallas** for execution.

Options are read from the encrypted file allocated to DD **LOGONDD**.

The file is copied as a text file (default transfer mode for standard files is text).

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	DD from which to read a script file. The script file is sent to the remote system for execution.
-encryptedfile	DD from which to read an encrypted command options file.
-host	Directs the command to a computer with a host name of **dallas**.
-stdin	Specification that the options following this one apply to the stdin file.
-encrypt	Specification that standard file data sent over the network is encrypted.
-compress	Specification for whether the standard file data transmitted across the network should be compressed.
-authenticate	Specification that standard file data sent over the network is authenticated.

Components

Universal Command Manager for z/OS

Universal Command Server for z/OS

Universal Copy

Copy from Remote zOS to Local zOS

Copy from Remote z/OS to Local z/OS (with Encryption, Compression, and Data Authentication) via Universal Copy

The following figure illustrates the copying of a file from a remote z/OS system to a local z/OS system (with encryption, compression, and data authentication).

```
//STEP1 EXEC UCMDPRC
//UNVOUT='DISP=SHR,DSN=MY.PDS(MEMBER)
//LOGONDD  DD DISP=SHR,DSN=MY.LOGON(USERID)
//SCRIPTDD DD *
/opt/universal/bin/ucopy < //'REMOTE.PDS(MEMBER)'
//SYSIN DD *
-script SCRIPTDD
-encryptedfile LOGONDD
-host dallas
-stdout -encrypt yes -compress yes -authenticate yes
/*
```

The JCL procedure **UCMDPRC** is used to execute the command. The command is sent to a remote system named **dallas** for execution. The **UNVOUT** DD specifies a local data set to use for the standard output of the remote command.

Options are read from the encrypted file allocated to DD **LOGONDD**.

The file is copied as a text file, since the default transfer mode for standard files is text.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	DD from which to read a script file. The script file is sent to the remote system for execution.
-encryptedfile	DD from which to read an encrypted command options file.
-host	Directs the command to a computer with a host name of **dallas**.
-stdout	Specification that the options following this one apply to the stdout file.
-encrypt	Specification that standard file data sent over the network is encrypted.
-compress	Specification for whether the standard file data transmitted across the network should be compressed.
-authenticate	Specification that standard file data sent over the network is authenticated.

Components

Universal Command Manager for z/OS

Universal Command Server for z/OS

Universal Copy

Copy from Local zOS to Remote Windows (with Windows Date Variables)

Copy from Local z/OS to Remote Windows (with Windows Date Variables) via Universal Copy

The following figure illustrates the copying of a file from a local z/OS system to a remote Windows system.

The file name on the Windows server is dynamically created based on the current date.

```
//S1 EXEC UCMDPRC
//UNVIN    DD DISP=SHR,DSN=hlq.input.file
//LOGONDD  DD DISP=SHR,DSN=hlq.userid(userid)
//SCRIPTDD DD *
@echo off
for /f "tokens=1 delims=/" %%a in ('date /t') do set daymm=%%a
for /f "tokens=2" %%a in ('echo %daymm%') do set mm=%%a
for /f "tokens=2 delims=/" %%a in ('date /t') do set dd=%%a
for /f "tokens=3 delims=/" %%a in ('date /t') do set yy=%%a
echo daymm:  %daymm%
echo mmddyy: %mm%%dd%%yy%
ucopy -output c:\temp\outputfile%mm%%dd%%yy%
//SYSIN    DD *
-script SCRIPTDD
-encryptedfile LOGONDD
-host dallas
/*
```

The JCL procedure **UCMDPRC** is used to execute the command. The command is sent to a remote system named **dallas** for execution. The -output option is used with the **ucopy** command to direct stdout to a local data set on the remote Windows server. The file name is created with a date variable. The date variable is set to the current date in the commands preceding the **ucopy** command.

Additional command line options are read from the encrypted file allocated to DD **LOGONDD**.

The file is copied as a text file, since the default transfer mode for standard files is text.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	DD from which to read a script file. The script file is sent to the remote system for execution.
-encryptedfile	DD from which to read an encrypted command options file.
-host	Directs the command to a computer with a host name of **dallas**.

Components

Universal Command Manager for z/OS

Universal Command Server for z/OS

Universal Copy

Copy from Local zOS to Remote UNIX (with UNIX Date Variables)

Copy from Local z/OS to Remote UNIX (with UNIX Data Variables) via Universal Copy

The following figure illustrates the copying of a file from a local z/OS system to a remote UNIX system. The file name on the UNIX server is dynamically created based on the current date.

```
//S1 EXEC UCMDPRC
//UNVIN    DD DISP=SHR,DSN=hlq.input.file
//LOGONDD  DD DISP=SHR,DSN=hlq.userid(userid)
//SCRIPTDD DD *
DATEN=`date +%d%m`
export DATEN
echo $DATEN
/opt/universal/bin/ucopy \
-output /tmp/output$DATEN.file
//SYSIN    DD *
-script SCRIPTDD
-encryptedfile LOGONDD
-host dallas
```

The JCL procedure **UCMDPRC** is used to execute the command. The command is sent to a remote system named **dallas** for execution. The stdout redirection character **>** is used with the **ucopy** command to direct stdout to a local data set on the remote server. The file name is created with a date variable, which is set to the current date in the commands preceding the **ucopy** command. The path to the **ucopy** binary must be specified if the directory is not defined in the user’s path environmental variable.

Additional command line options are read from the encrypted file allocated to DD **LOGONDD**. The UNIX continuation character **** is used to split the **ucopy** command to two lines.

The file is copied as a text file, since the default transfer mode for standard files is text.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	DD from which to read a script file. The script file is sent to the remote system for execution.
-encryptedfile	DD from which to read an encrypted command options file.
-host	Directs the command to a computer with a host name of **dallas**.

Components

Universal Command Manager for z/OS

Universal Command Server for z/OS

Universal Copy

Copy from Remote UNIX to Local zOS Using cat Command

Copy from Remote UNIX to Local z/OS Using UNIX cat Command via Universal Command Manager for z/OS

The following figure illustrates the copying of a file from a remote UNIX system to a local z/OS system using the UNIX **cat** command.

```
//UNIXCAT  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*
//         JCLLIB ORDER=username.UNV.SUNVSAMP
//*
//STEP1    EXEC UCMDPRC,
//               STDOUT='DISP=SHR,DSN=username.UNIX.FILE1'
//SYSIN    DD  *
-cmd 'cat /export/home/username/file1'
-host Unix_1 -userid username -pwd password -level audit
/*
```

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-cmd	Remote command **`cat /export/home/username/file1`** to execute. The **cat** program copies the files specified on the command line to its **stdout.n**.
-host	Directs the command to a computer with host address **Unix_1**.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-level	Message level output for this command execution.

Components

Universal Command Manager for z/OS

Copy from Remote UNIX to Local Windows

Copy from Remote UNIX to Local Windows via Universal Command Manager for Windows

The following figure illustrates the copying of a file from a remote UNIX system to a local Windows system. Although the command is shown on two lines, it should be entered on one line at the command prompt. If it is coded in a script, the Windows continuation character of **¬** must be used.

```
ucmd –cmd “ /opt/universal/bin/ucopy unixinputfile”
-host unixhost -encryptedfile unixid.file > c:\temp\ntoutputfile
```

The standard out of the **ucopy** command on the remote host is redirected back to the local host and written to **`c:\temp\ntoutput.file`**. The command **ucopy** is installed as part of Universal Command Server on the remote system.

The file is copied as a text file, since the default transfer mode for standard files is text.

Parameters

The following parameters should be changed to match your information:

Parameter	Description
ntoutputfile	Path and filename of output file
unixinputfile	Path and file name of input file
unixhost	IP address of remote UNIX server
unixid.file	File on the Windows server used to house the authentication parameters for the UNIX server

Command Line Options

The command line options used in this example are:

Option	Description
-cmd	Remote command **ucopy file** to execute. The **ucopy** program copies the files specified on the command line to its STDOUT.
-host	Directs the command to a computer with a host name of **unixhost**.
-encryptedfile	File from which to read encrypted command options.

Components

Universal Command Manager for Windows

Universal Copy

Copy From Local Windows to Remote UNIX

Copy From Local Windows to Remote UNIX via Universal Command Manager for Windows

The following figure illustrates the copying of a file from a local Windows system to a remote UNIX system. Although the command is shown on two lines, it should be entered on one line at the command prompt. If it is coded in a script, the Windows continuation character of **¬** must be used.

```
ucmd –cmd “ /opt/universal/bin/ucopy -output /tmp/unixoutputfile”
-host unixhost -encryptedfile unixid.file
< c:\temp\ntinputfile
```

The stdin of the **ucmd** manager on the local host is redirected to the stdout of the remote host and written to **`/tmp/unixoutputfile`**. The command **ucopy** is installed as part of Universal Command Server on the remote system. The file is copied as a text file since the default transfer mode for standard files is text.

Parameters

The following parameters should be changed to match your information:

Parameter	Description
unixoutputfile	Path and filename of output file
ntinputfile	Path and file name of input file
unixhost	IP address of sending UNIX server
unixid.file	File on the Windows server used to house the authentication parameters for the UNIX server

Command Line Options

The command line options used in this example are:

Option	Description
-cmd	Remote command **ucopy file** to execute. The **ucopy** program copies the files specified on the command line to its stdout.
-host	Directs the command to a computer with a host name of **unixhost**.

-encryptedfile	File from which to read encrypted command options.

Components

Universal Command Manager for Windows

Universal Copy

Copy from Remote UNIX to Local Windows Using the UNIX cat Command

Copy from Remote UNIX to Local Windows Using the UNIX cat Command via Universal Command Manager for Windows

The following figure illustrates copying of file from a remote UNIX system to a local Windows system using the UNIX **cat** command.

Although the command is shown on two lines, it should be entered as one line at the command prompt.

```
ucmd -cmd "cat ~/file" -host dallas
     -userid joe –pwd password -comment "copy ~/file from dallas" > localfile
```

The stdout of the **cat** command on the remote host is redirected back to the local host and written to the stdout of **ucmd**, which is then redirected to the local file **localfile**.

Command Line Options

The command line options used in this example are:

Option	Description
-cmd	Remote command - **"cat ~/file"** - to execute. The **cat** program copies the files specified on the command line to its stdout.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-comment	Description of the process executed by Universal Command.

The file is copied as a text file, since the default transfer mode is **text**.

Components

Universal Command Manager for UNIX

Universal Copy

Copy from Local UNIX to Remote Windows

Copy from Local UNIX to Remote Windows via Universal Command Manager for UNIX

The following figure illustrates the copying of a file from a local UNIX system to a remote Windows system.

Although the command is shown on two lines, it should be entered as one line at the command prompt.

```
ucmd –cmd 'ucopy > remotefile' -host dallas
     -userid joe –pwd password < localfile
```

The **ucopy** command receives its stdin file from ucmd. The standard in of UCMD is redirected from **localfile**.

The command **ucopy** is installed as part of Universal Command Server on the remote system.

The file is copied as a text file, since the default transfer mode is **text**.

Command Line Options

The command line options used in this example are:

Option	Description
-cmd	Remote command **`ucopy > remotefile`** to execute. The **ucopy** program copies its standard in to its standard out.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

Components

Universal Command Manager for UNIX

Universal Copy

Copy Encrypted File from Local UNIX to Remote Windows

Copy Encrypted File from Local UNIX to Remote Windows via Universal Command Manager for UNIX

The following figure illustrates the copying of a file from a local UNIX system to a remote Windows server.

Although the command is shown on two lines, it should be entered on one line at the command prompt. If it is coded in a script, the UNIX continuation character of \ must be used.

```
ucmd –cmd 'ucopy –output c:\temp\ntoutput.file' -host nthost
     -encryptedfile login.file < /tmp/unixinput.file
```

The stdin of the ucmd manager on the local host is redirected to the remote host and written to stdout file **`c:\temp\ntoutput.file`**. The command **ucopy** is installed as part of Universal Command Server on the remote system.

The file is copied as a text file, since the default transfer mode for standard files is text.

Command Line Options

The command line options used are:

Option	Description
-cmd	Remote command **`ucopy file`** to execute. The **ucopy** program copies the files specified on the command line to its stdout.
-host	Directs the command to a computer with a host name of **nthost**.
-encryptedfile	File from which to read encrypted command options.

Components

Universal Command Manager for UNIX

Universal Copy

Copy from Remote Windows to Local UNIX

Copy from Remote Windows to Local UNIX via Universal Command Manager for UNIX

The following figure illustrates the copying of a file from a remote Windows system to a local UNIX system.

Although the command is shown on two lines, it should be entered as one line at the command prompt.

```
ucmd –cmd 'ucopy file' -host dallas
     -userid joe –pwd password > localfile
```

The stdout of the **ucopy** command on the remote host is redirected back to the local host and written to the stdout of ucmd, which is then redirected to the local file **localfile**.

The command **ucopy** is installed as part of UCMD Server on the remote system.

The file is copied as a text file since the default transfer mode is **text**.

Command Line Options

The command line options used in this example are:

Option	Description
-cmd	Remote command **`ucopy file`** to execute. The **ucopy** program copies the files specified on the command line to its stdout.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

Components

Universal Command Manager for Windows

Universal Copy

Copy Encrypted File from Remote Windows to Local UNIX

Copy Encrypted File from Remote Windows to Local UNIX via Universal Command Manager for UNIX

The following figure illustrates the copying of a file from a remote Windows system to a local UNIX server.

If it is coded in a script, then the UNIX continuation character of \ must be used.

```
ucmd –cmd 'ucopy ntinputfile' -host nthost -encryptedfile
ntid.file > /tmp/unixoutputfile
```

The stdout of the **ucopy** command on the remote host is redirected back to the local host and written to **`/tmp/unixoutputfile`**. The command **ucopy** is installed as part of Universal Command Server on the remote system.

The file is copied as a text file, since the default transfer mode for standard files is text.

Command Line Options

The command line options used are:

Option	Description
-cmd	Remote command **`ucopy file`** to execute. The **ucopy** program copies the files specified on the command line to its stdout.
-host	Directs the command to a computer with a host name of **nthost**.
-encryptedfile	File from which to read encrypted command options.

Components

Universal Command Manager for Windows

Universal Copy

Copy from Remote Windows to Local IBM i via UCMD Manager

Copy from Remote Windows to Local IBM i via Universal Command Manager for IBM i

The following figures illustrate copying a file from a remote Windows system to a local file.

```
STRUCM CMD('ucopy infile') HOST(dallas) USERID(joe) PWD(password) SOTFILE(localfile)
```

UCOPY, which the UCMD Server invokes on system **dallas**, retrieves data from the file named **infile**. It sends data to the UCMD Manager running under IBM i using standard output. The UCMD Manager, in turn, receives input via stdin and writes to file specified by SOTFILE, **localfile**. The file is copied via UCMD as a text file, since the default transfer mode is **text**.

```
STRUCM CMD('ucopy c:\ntinput.file') HOST(nthost) USERID(joe) PWD(akkSdiq) SOTFILE(library/outputfile)
SOTMBR(member)
```

UCOPY runs on the remote host and retrieves data from **`c:\ntinput.file`**. UCOPY output is redirected back to the local host and written to the stdout of **STRUCM**. **STRUCM** output is, in turn, directed to the local file **SOTFILE** and, optionally, **SOTMBR**. The file is copied via UCMD as a text file, since the default transfer mode for standard files is text.

The command **ucopy** is installed as part of UCMD Server on the remote system.

Command Line Options

The command line options used are:

Option	Description
CMD	Remote command to execute. The **ucopy** program copies the files specified on the command line to its stdout.
HOST	Directs the command to a computer with a host name of **nthost**.
USERID	Remote user ID with which to execute the command.
PWD	Password for the user ID.
SOTFILE [SOTMBR]	Location to which the stdout file data is written.

Components

Universal Command Manager for IBM i

Universal Command Server for Windows

Universal Copy

Copy from Remote IBM i to Local Windows via UCMD Manager

Copy from Remote IBM i to Local Windows via Universal Command Manager for IBM i

The following figure illustrates the copying of a file initiated by Windows, which copies the first member of a file from IBM i to a file on the Windows system.

```
ucmd -host sysName -userid userId –pwd password
-cmd "strucp frmfile(mylib/myfile)" > D:\tmp\File400.txt
```

UCMD running on Windows invokes STRUCP via a UCMD Server running on IBM i. The FRMFILE parameter overrides input from stdin to the file **mylib/file**. Since the FRMMBR parameter is not used, input defaults to the file member ***FIRST**. Data is transferred from **mylib/myfile** to **D:\tmp\File400.txt** via UCMD Manager stdout.

The command **STRUCP** is installed as part of UCMD Server on the IBM i system.

The file is copied as a text file, since the default transfer mode is **text**.

Command Line Options

The command line options used in this example are:

Option	Description
-cmd	Remote command to execute on the IBM i.
-host	Directs the command to a computer with a host name of **sysName**.
-userid	IBM i user ID with which to execute the command.
-pwd	Password for the user ID.

Components

Universal Command Manager for Windows

Universal Command Server for IBM i

Universal Copy

Copy from Local Windows to Remote IBM i via UCMD Manager

Copy from Local Windows to Remote IBM i

The following figure illustrates the copying of a file initiated by Windows which copies a file from Windows to the first member of a file on IBM i.

```
ucmd -host sysName -userid userId -pwd password
-cmd "strucp tofile(mylib/readme)" < D:\tmp\README.txt
```

Using redirected stdin, UCMD Manager, running under Windows sends, transfers data to a UCMD Server running on the remote IBM i system, **sysName**. The UCMD Server on **sysName** invokes UCOPY to transfer the data to **`mylib/readme`**. **`mylib/readme`** file member ***FIRST** is used since TOMBR was not specified.

The command **STRUCP** is installed as part of UCMD Server on the IBM i system.

The file is copied as a text file since the default transfer mode is **text**.

Command Line Options

The command line options used in this example are:

Option	Description
-cmd	Remote command to execute on the IBM i.
-host	Directs the command to a computer with a host name of **sysName**.
-userid	IBM i user ID with which to execute the command.
-pwd	Password for the user ID.

Components

Universal Command Manager for Windows

Universal Command Server for IBM i

Universal Copy

Copy from Local IBM i to Remote Windows via UCMD Manager

Copy from Local IBM i to Remote Windows via Universal Command Manager for IBM i

The following figures illustrate copying a file from a local IBM i system to a remote Windows system.

```
STRUCM CMD('ucopy > remotefile') HOST(dallas)
       USERID(joe) PWD(password) SINFILE(localfile)
```

The UCOPY utility running on the remote Windows system receives its standard in file from STRUCM via the Windows agent. The standard in of STRUCM is read from **localfile**, as specified by **SINFILE**.

The file is copied via UCMD as a text file, since the default transfer mode is **text**.

```
STRUCM CMD('ucopy -output c:\ntoutput.file') HOST(nthost) USERID(joe) PWD(akkSdiq)
SINFILE(library/inputfile) SINMBR(member)
```

The **ucopy** command receives its stdin file from STRUCM. The stdin of STRUCM is redirected from **SINFILE** and, optionally, **SINMBR** to stdout of **ucopy**, via the Windows agent.
UCOPY stdout is redirected to `c:\ntoutput.file` using the UCOPY **output** parameter.

The file is copied via UCMD as a text file, since the default transfer mode for standard files is text.

The command **ucopy** is installed as part of UCMD Server on the remote system.

Command Line Options

The command line options used are:

Option	Description
CMD	Remote command **`ucopy > file`** to execute. The **ucopy** program copies its stdin to its stdout.
HOST	Directs the command to a computer with a host name of **nthost**.
USERID	Remote user ID with which to execute the command.
PWD	Password for the user ID.
SINFILE [SINMBR]	Location from which the stdin file data is written.

Components

Universal Command Manager for IBM i

Universal Command Server for Windows

Universal Copy

Copy from Remote Windows to Local HP NonStop via UCOPY

Copy from Remote Windows to Local HP NonStop via Universal Copy

The following figure illustrates the copying of a file from a remote Windows system to a local file.

Although the command shown is on two lines, it should be entered on one line at the command prompt.

The HP NonStop manager is executed within the TACL environment.

```
run $SYSTEM.UNVBIN.ucmd /OUT outputfile/ –cmd 'ucopy inputfile' -host dallas -userid joe –pwd akkSdiq
```

The stdout of the **ucopy** command on the remote host is redirected back to the local host and written to the stdout of ucmd, which is then redirected to the local file **outputfile**. The command **ucopy** is installed as part of Universal Command Server on the remote system.

The file is copied as a text file since the default transfer mode for standard files is text.

Command Line Options

The command line options used are:

Option	**Description**
-cmd	Remote command **ucopy file** to execute. The **ucopy** program copies the files specified on the command line to its stdout.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

Components

Universal Command Manager for HP NonStop

Universal Command Server for Windows

Universal Copy

Copy from Local HP NonStop to Remote Windows via UCOPY

Copy from Local HP NonStop to Remote Windows via Universal Copy

The following figure illustrates the copying of a local file to a remote Windows system.

Although the command is shown on two lines, it should be entered on one line at the command prompt.

The HP NonStop manager is executed within the TACL environment.

The file is copied as a text file, since the default transfer mode for standard files is text.

```
run $SYSTEM.UNVBIN.ucmd /IN inputfile/ –cmd 'ucopy –output outputfile'
-host dallas -userid joe –pwd akkSdiq
```

The **ucopy** command receives its stdin file from ucmd. The stdin of ucmd is redirected from **inputfile**. The command **ucopy** is installed as part of Universal Command Server on the remote system.

Command Line Options

The command line options used are:

Option	Description
-cmd	Remote command **ucopy** to execute. The **ucopy** program copies stdin to stdout.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

Components

Universal Command Manager for HP NonStop

Universal Command Server for Windows

Universal Copy

Copy from Remote Windows to Local HP NonStop (using STDOUT) - 1

Copy from Remote Windows to Local HP NonStop via UCMD Manager for HP NonStop

The following figure illustrates the copying of a file from a remote Windows system to a local file.

Although the command is shown on two lines, it should be entered on one line at the command prompt.

```
run $SYSTEM.UNVBIN.ucmd –cmd 'ucopy file' -host dallas -userid joe
-pwd password –stdout –localfile localfile
```

The standard out of the **ucopy** command on the remote host is redirected back to the local host and written to the standard out of UCMD, which then is redirected to the local file **localfile**. The command **ucopy** is installed as part of UCMD Server on the remote system. The process will authenticate and run under the authority of userid **joe**.

Command Line Options

The command line options used in this example are:

Option	Description
-cmd	Remote command **ucopy file** to execute.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-stdout	Start of the stdout options.
-localfile	Filename to which to redirect output.

Components

Universal Command Manager for HP NonStop

Universal Command Server for Windows

Universal Copy

Copy from Remote Windows to Local HP NonStop (using STDOUT) - 2

Copy from Remote Windows to Local HP NonStop via UCMD Manager for HP NonStop

The following figure illustrates the copying of a file from a remote Windows system to a local file.

Although the command is shown on two lines, it should be entered on one line at the command prompt.

```
run ucmd –cmd 'ucopy file' -host dallas –server “ –script_type OSS”
         -userid joe –pwd password –stdout –localfile localfile
```

The stdout of the **ucopy** command on the remote host is redirected back to the local host and written to the standard out of UCMD, which is then redirected to the local file **localfile**.

The command **ucopy** is installed as part of UCMD Server on the remote system.

The file is copied as a text file, since the default transfer mode is **text**.

Command Line Options

The command line options used in this example are:

Option	Description
-cmd	Remote command **ucopy file** to execute. The **ucopy** program copies the files specified on the command line to its stdout.
-host	Directs the command to a computer with a host name of **dallas**.
-server	Command lines options for the UCMD Server process. The value **-script_type OSS** is specified to notify the UCMD Server that it is to execute an OSS process, since Universal Copy is a native OSS program.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-stdout	Start of the stdout options.
-localfile	Filename to which to redirect output.

Components

Universal Command Manager for HP NonStop

Universal Command Server for Windows

Universal Copy

Copy from Local HP NonStop to Remote Windows (using STDIN)

Copy from Local HP NonStop to Remote Windows via UCMD Manager for HP NonStop

The following figure illustrates the copying of a file from a local HP NonStop system to a remote Windows system.

Although the command is shown on two lines, it should be entered on one line at the command prompt.

```
run $SYSTEM.UNVBIN.ucmd –cmd 'ucopy > remotefile' -host dallas -userid joe –pwd password -stdin
–localfile localfile
```

The **ucopy** command receives its standard in file from the UCMD **localfile** parameter. The file is written to **remotefile** on the remote system. The command **ucopy** is installed as part of UCMD Server on the remote system. The process will authenticate and run under the authority of userid **joe**.

Command Line Options

The command line options used in this example are:

Option	Description
-cmd	Remote command **ucopy file** to execute.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-stdin	Start of the stdin options.
-localfile	Filename from which to redirect input.

Components

Universal Command Manager for HP NonStop

Universal Command Server for Windows

Universal Copy

Copy from Local HP NonStop to Remote Windows (using STDOUT)

Copy from Local HP NonStop to Remote Windows via UCMD Manager for HP NonStop

The following figure illustrates the copying of a file from a local HP NonStop system to a remote Windows system.

Although the command is shown on multiple lines, it should be entered as one line at the command prompt.

```
run ucmd –cmd 'ucopy > remotefile' -host dallas
         –server “ –script_type OSS” -userid joe –pwd password
         -stdin –localfile localfile
```

The **ucopy** command receives its standard in file from UCMD. The standard in of UCMD is redirected from **localfile**.

The command **ucopy** is installed as part of UCMD Server on the remote system.

The file is copied as a text file since the default transfer mode is **text**.

Command Line Options

The command line options used in this example are:

Option	Description
-cmd	Remote command **ucopy > remotefile** to execute. The **ucopy** program copies it stdin to its stdout.
-host	Directs the command to a computer with a host name of **dallas**.
-server	Command lines options for the UCMD Server process. The value **-script_type OSS** is specified to notify the UCMD Server that it is to execute an OSS process, since universal copy is a native OSS program.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
stdin	Start of the stdout options.
-localfile	Filename to which to redirect output.

Components

Universal Command Manager for HP NonStop

Universal Command Server for Windows

Universal Copy

Indesca - 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

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

Overview

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. 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 (see Universal Encrypt).

> ⚠ **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, 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.

z/OS	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): `PARM='ENVAR("UCMDLEVEL=INFO")/'`

UNIX	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 be a called program.
	Example of setting an environment variable (set option UCMDLEVEL to a value of INFO in a bourne, bash, or korn shell):
	`UCMDLEVEL=INFO`
	`export UCMDLEVEL`

Windows	Environment variables in Windows are defined as part of the Windows console command environment. As such, console commands are used to set environment variables. Example of setting an environment variable (set option UCMDLEVEL to a value of INFO): `SET UCMDLEVEL=INFO`
IBM i	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): `ADDENVVAR ENVVAR(UCMDLEVEL) VALUE(INFO)`

HP NonStop	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): `PARAM UCMDLEVEL INFO`

Configuration Methods - Configuration File

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).

z/OS	Configuration files are members of a PDSE. The data set record format is fixed or fixed block with a record length of 80. No line numbers can exist in columns 72-80. All 80 columns are processed as data. All configuration files are installed in the **UNVCONF** library. See Configuration File Syntax, below, for the configuration file syntax.
UNIX	Configuration files are regular text files on UNIX. The files can be edited with a text editor. See Configuration File Syntax, below, for the configuration file syntax. 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. • /etc/opt/universal • /etc/universal (installation default) • /etc/stonebranch (obsolete as of version 2.2.0) • /etc • /usr/etc/universal • /usr/etc/stonebranch (obsolete as of version 2.2.0) • /usr/etc
Windows	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. Universal Configuration Manager provides a graphical interface and context-sensitive help, and helps protect the integrity of the configuration file by validating all changes to configuration option values.
IBM i	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 Configuration File Syntax, below, for the configuration file syntax.
HP NonStop	The configuration files on HP NonStop are stored as EDIT files, file code 101, within the **$SYSTEM.UNVCONF** subvolume. The files can be edited with the EDIT editor. See Configuration File Syntax, below, for the configuration file syntax.

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

Indesca Configuration Options

The following configuration options are available for Indesca 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 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

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 be 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.

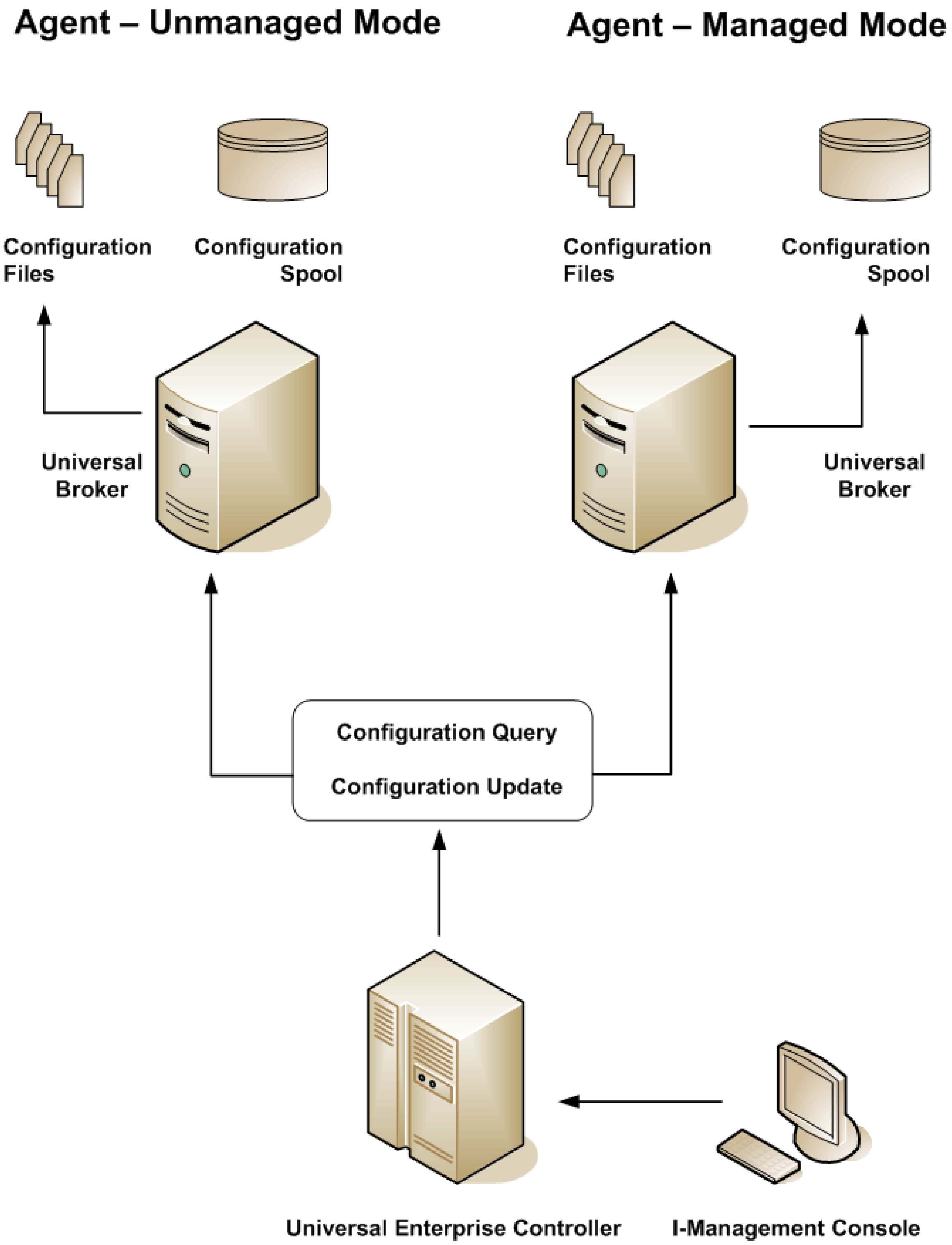
- If a configuration file exists in the configuration directory, it is overwritten.
- If a configuration file does not exist, it is created.

⚠ **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.



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

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.

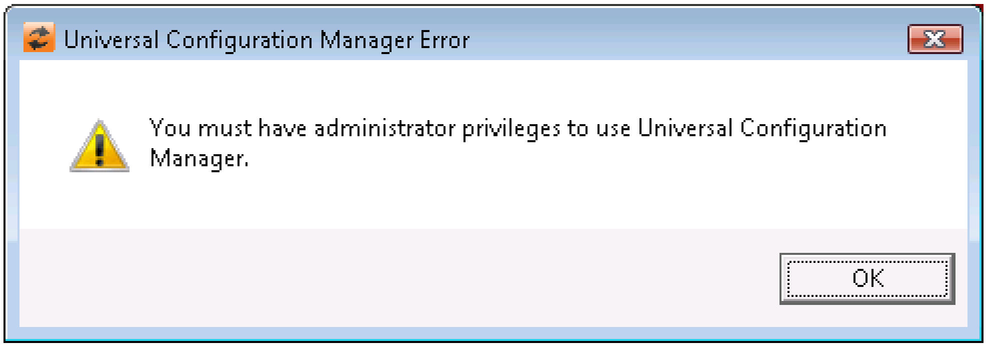
It is available to all user accounts in the Windows Administrator group.

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.

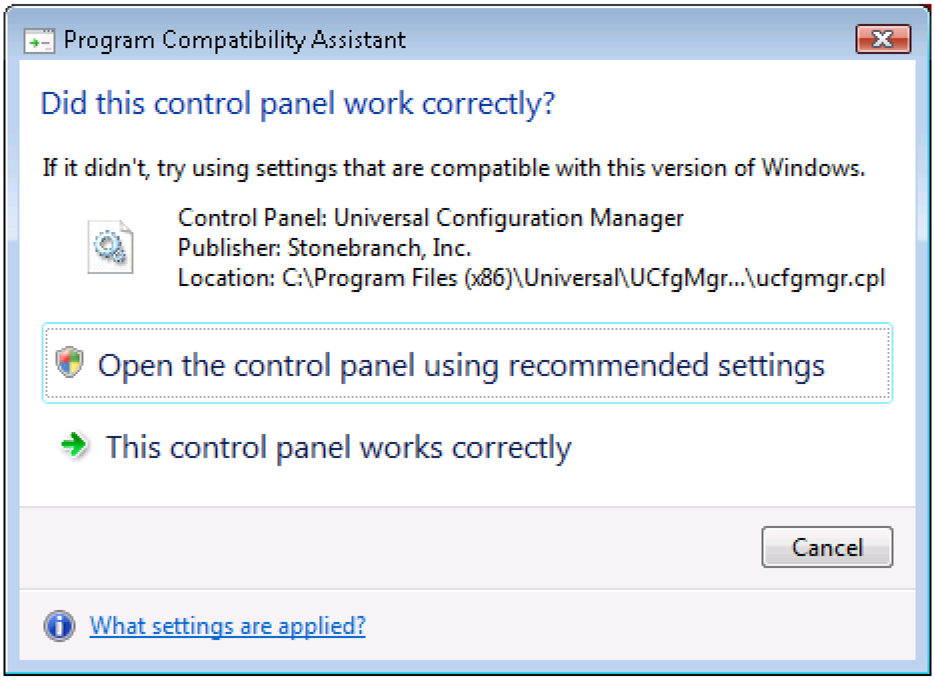
With these two features enabled, the expected Universal Configuration Manager behavior is as follows:

1. Universal Configuration Manager may issue the following error.


Universal Configuration Manager Error

You must have administrator privileges to use Universal Configuration Manager.

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


Program Compatibility Assistant

Did this control panel work correctly?

If it didn't, try using settings that are compatible with this version of Windows.

Control Panel: Universal Configuration Manager
Publisher: Stonebranch, Inc.
Location: C:\Program Files (x86)\Universal\UCfgMgr...\ucfgmgr.cpl

Open the control panel using recommended settings
This control panel works correctly
Cancel
What settings are applied?

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:

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

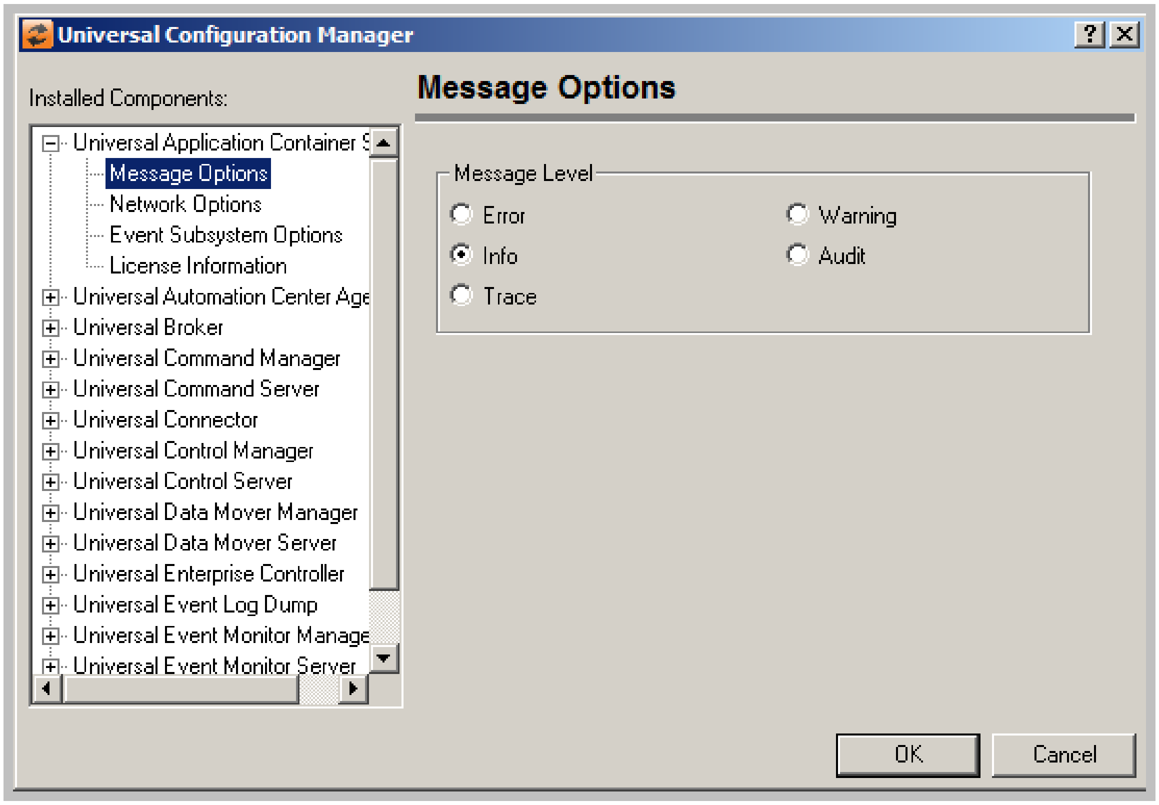
Windows XP, Windows Vista, Windows 7, Windows Server 2008 / 2008 R2
Newer versions of Windows support a Control Panel view that places applet icons within categories. This "category view" may affect the location of the Universal Configuration Manager icon.

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.


Universal Configuration Manager
Installed Components:
Universal Application Container S
Message Options
Network Options
Event Subsystem Options
License Information
Universal Automation Center Age
Universal Broker
Universal Command Manager
Universal Command Server
Universal Connector
Universal Control Manager
Universal Control Server
Universal Data Mover Manager
Universal Data Mover Server
Universal Enterprise Controller
Universal Event Log Dump
Universal Event Monitor Manage
Universal Event Monitor Server
OK
Cancel

Message Options

Message Level

○ Error ○ Warning
● Info ○ Audit
○ Trace

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:

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

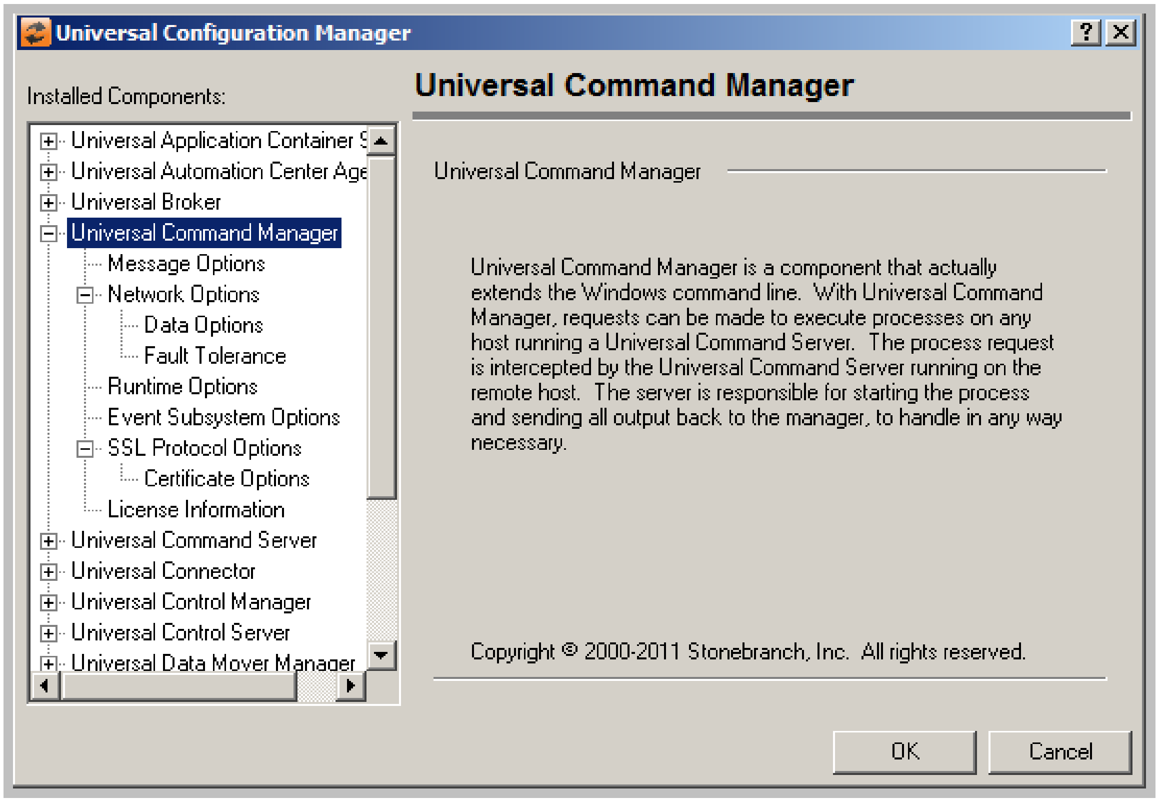
Universal Command Installed Components

Universal Command Manager

The following figure illustrates the Universal Configuration Manager screen for the Universal Command Manager.

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

The text describes the selected component, Universal Command Manager.



Universal Command Manager is a component that actually extends the Windows command line. With Universal Command Manager, requests can be made to execute processes on any host running a Universal Command Server. The process request is intercepted by the Universal Command Server running on the remote host. The server is responsible for starting the process and sending all output back to the manager, to handle in any way necessary.

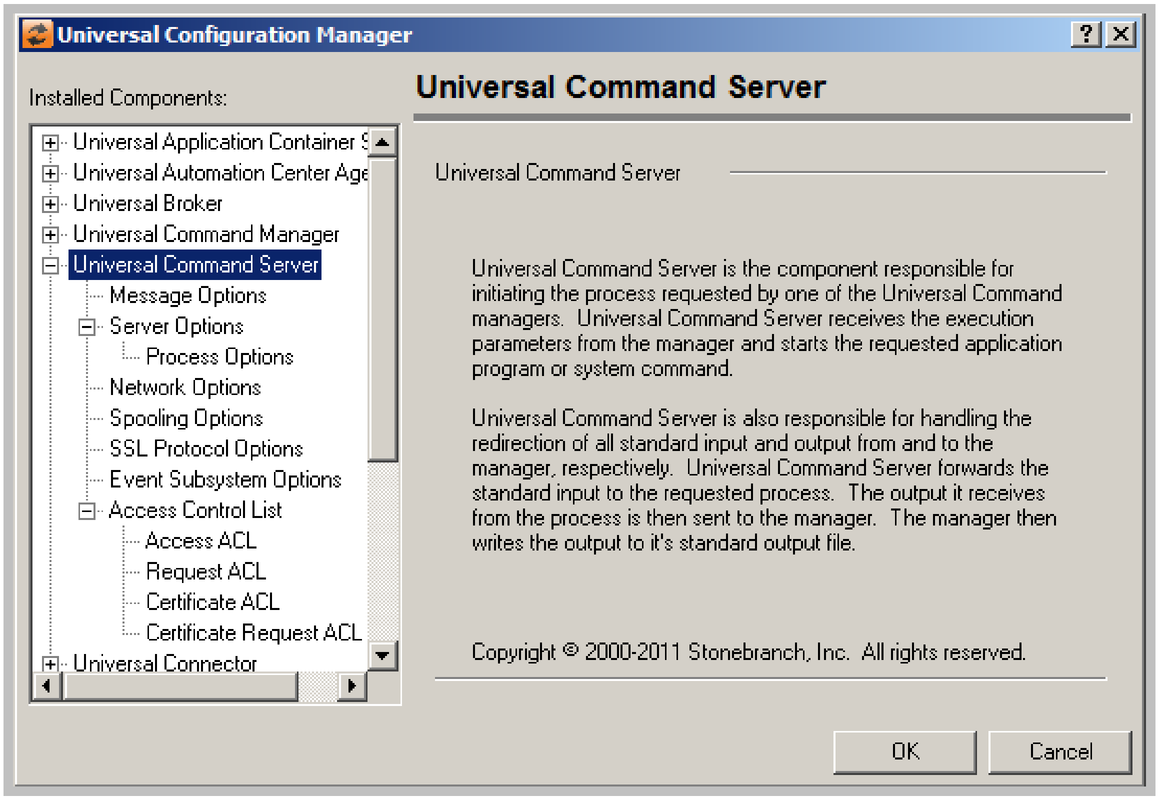
Copyright © 2000-2011 Stonebranch, Inc. All rights reserved.

Universal Command Server

The following figure illustrates the Universal Configuration Manager screen for the Universal Command Server.

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

The text describes the selected component, Universal Command Server.



Universal Command Server

Universal Command Server

Universal Command Server is the component responsible for initiating the process requested by one of the Universal Command managers. Universal Command Server receives the execution parameters from the manager and starts the requested application program or system command.

Universal Command Server is also responsible for handling the redirection of all standard input and output from and to the manager, respectively. Universal Command Server forwards the standard input to the requested process. The output it receives from the process is then sent to the manager. The manager then writes the output to it's standard output file.

Copyright © 2000-2011 Stonebranch, Inc. All rights reserved.

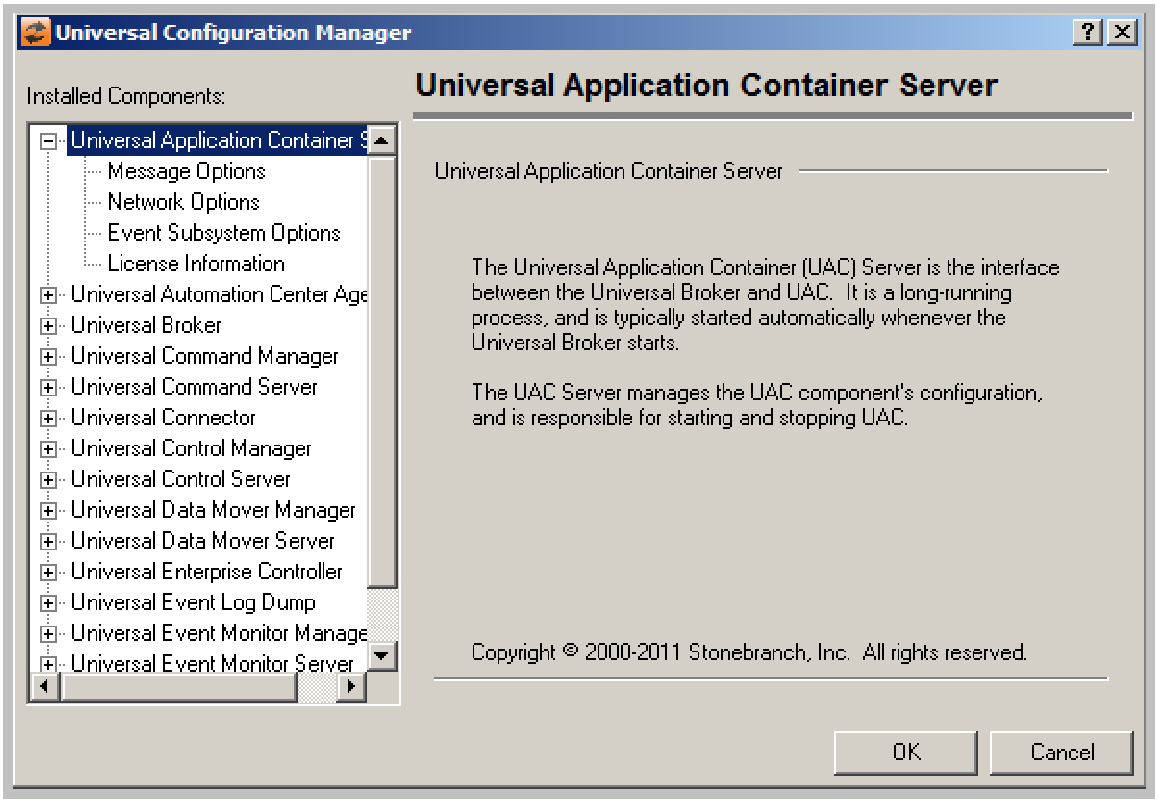
Universal Command Agent for SOA Installed Components

Universal Application Container Server

The following figure illustrates the Universal Configuration Manager screen for the Universal Application Container Server.

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

The text describes the selected component, Universal Application Container Server.



Universal Configuration Manager
Installed Components:
Universal Application Container S
Message Options
Network Options
Event Subsystem Options
License Information
Universal Automation Center Age
Universal Broker
Universal Command Manager
Universal Command Server
Universal Connector
Universal Control Manager
Universal Control Server
Universal Data Mover Manager
Universal Data Mover Server
Universal Enterprise Controller
Universal Event Log Dump
Universal Event Monitor Manage
Universal Event Monitor Server
OK
Cancel

Universal Application Container Server

Universal Application Container Server

The Universal Application Container (UAC) Server is the interface between the Universal Broker and UAC. It is a long-running process, and is typically started automatically whenever the Universal Broker starts.

The UAC Server manages the UAC component's configuration, and is responsible for starting and stopping UAC.

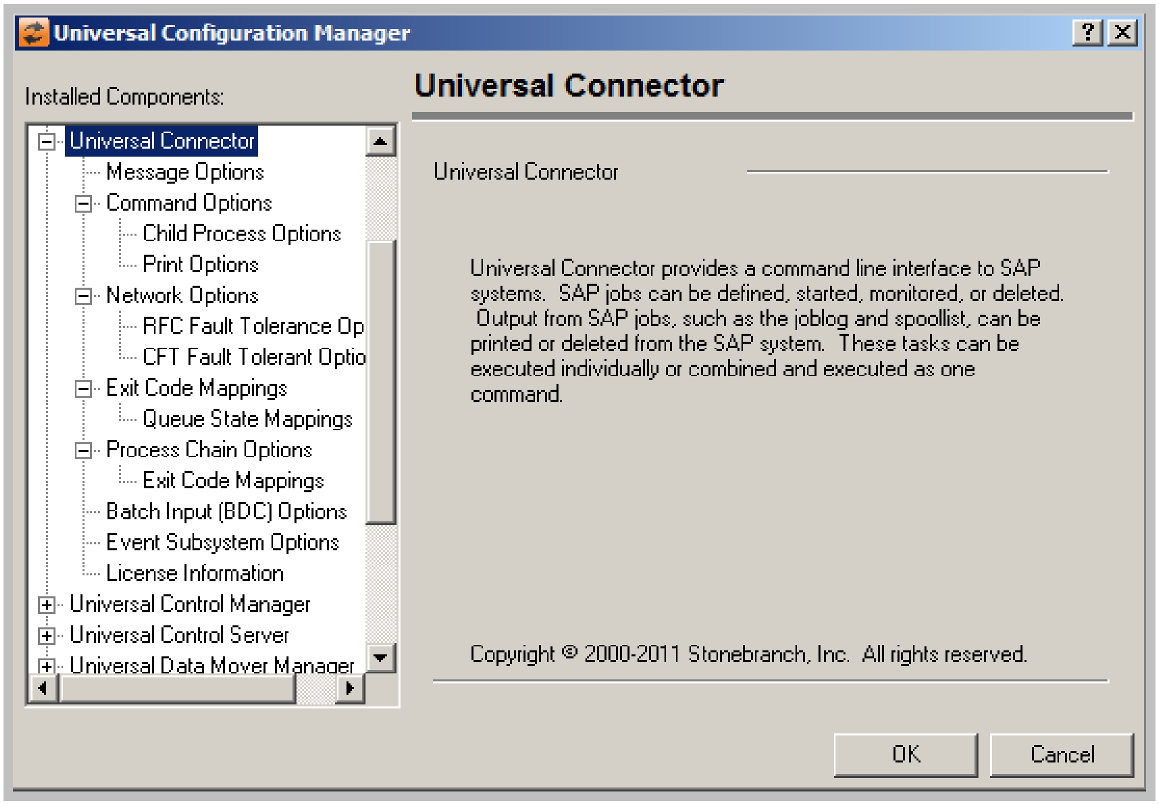
Copyright © 2000-2011 Stonebranch, Inc. All rights reserved.

Universal Connector Component

The following figure illustrates the Universal Configuration Manager screen for the Universal Connector.

The Installed Components list identifies all of the Universal Connector property pages.

The text describes the selected component, Universal Connector.



Universal Connector

Universal Connector

Universal Connector provides a command line interface to SAP systems. SAP jobs can be defined, started, monitored, or deleted. Output from SAP jobs, such as the joblog and spoollist, can be printed or deleted from the SAP system. These tasks can be executed individually or combined and executed as one command.

Copyright © 2000-2011 Stonebranch, Inc. All rights reserved.

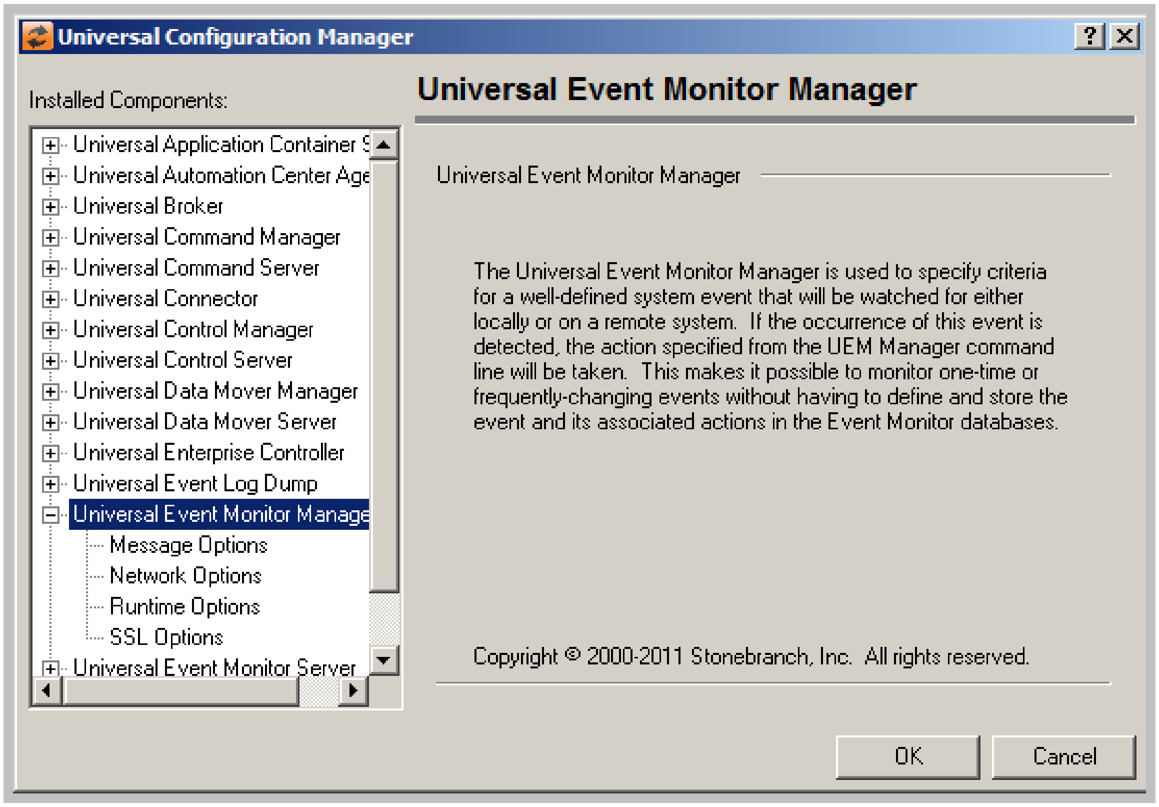
Universal Event Monitor Installed Components

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 Configuration Manager
Installed Components:
Universal Application Container S
Universal Automation Center Age
Universal Broker
Universal Command Manager
Universal Command Server
Universal Connector
Universal Control Manager
Universal Control Server
Universal Data Mover Manager
Universal Data Mover Server
Universal Enterprise Controller
Universal Event Log Dump
Universal Event Monitor Manage
Message Options
Network Options
Runtime Options
SSL Options
Universal Event Monitor Server
Universal Event Monitor Manager
Universal Event Monitor Manager
OK
Cancel

The Universal Event Monitor Manager is used to specify criteria for a well-defined system event that will be watched for either locally or on a remote system. If the occurrence of this event is detected, the action specified from the UEM Manager command line will be taken. This makes it possible to monitor one-time or frequently-changing events without having to define and store the event and its associated actions in the Event Monitor databases.

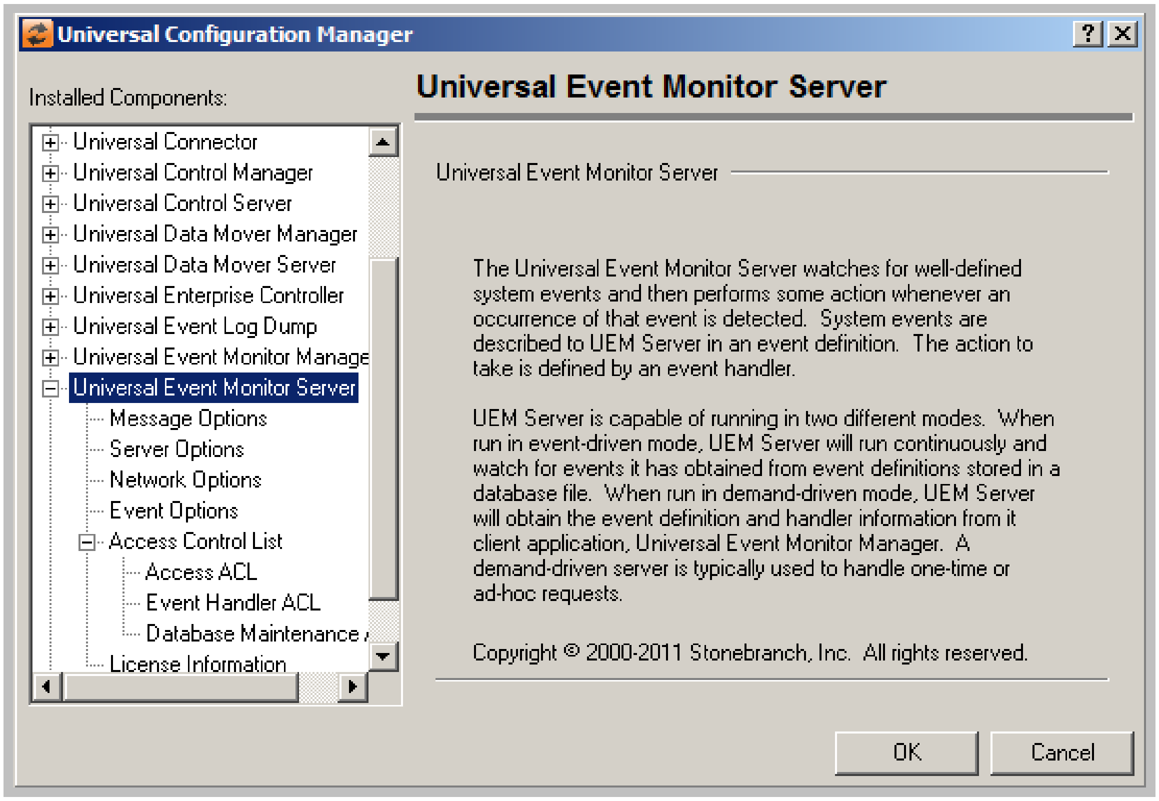
Copyright © 2000-2011 Stonebranch, Inc. All rights reserved.

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 Configuration Manager
Installed Components:
Universal Connector
Universal Control Manager
Universal Control Server
Universal Data Mover Manager
Universal Data Mover Server
Universal Enterprise Controller
Universal Event Log Dump
Universal Event Monitor Manage
Universal Event Monitor Server
Message Options
Server Options
Network Options
Event Options
Access Control List
Access ACL
Event Handler ACL
Database Maintenance
License Information
Universal Event Monitor Server
Universal Event Monitor Server
OK
Cancel

The Universal Event Monitor Server watches for well-defined system events and then performs some action whenever an occurrence of that event is detected. System events are described to UEM Server in an event definition. The action to take is defined by an event handler.

UEM Server is capable of running in two different modes. When run in event-driven mode, UEM Server will run continuously and watch for events it has obtained from event definitions stored in a database file. When run in demand-driven mode, UEM Server will obtain the event definition and handler information from it client application, Universal Event Monitor Manager. A demand-driven server is typically used to handle one-time or ad-hoc requests.

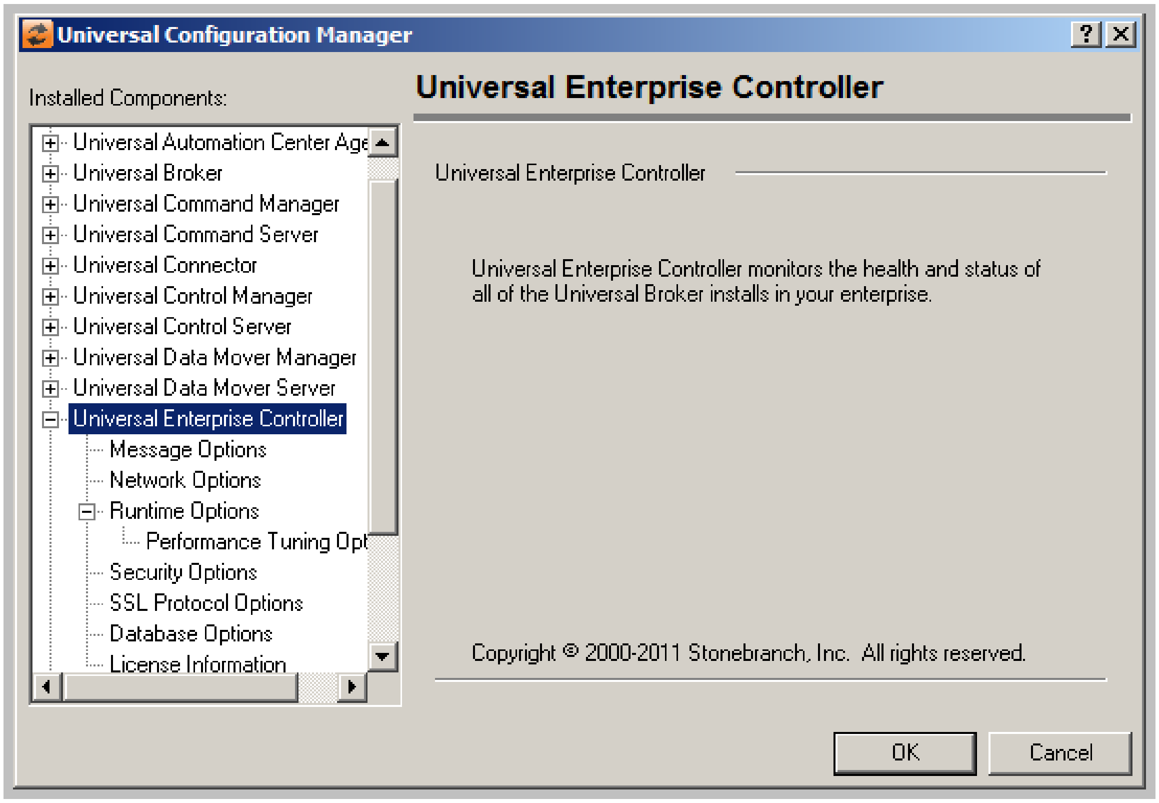
Copyright © 2000-2011 Stonebranch, Inc. All rights reserved.

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 Enterprise Controller

Universal Enterprise Controller

Universal Enterprise Controller monitors the health and status of all of the Universal Broker installs in your enterprise.

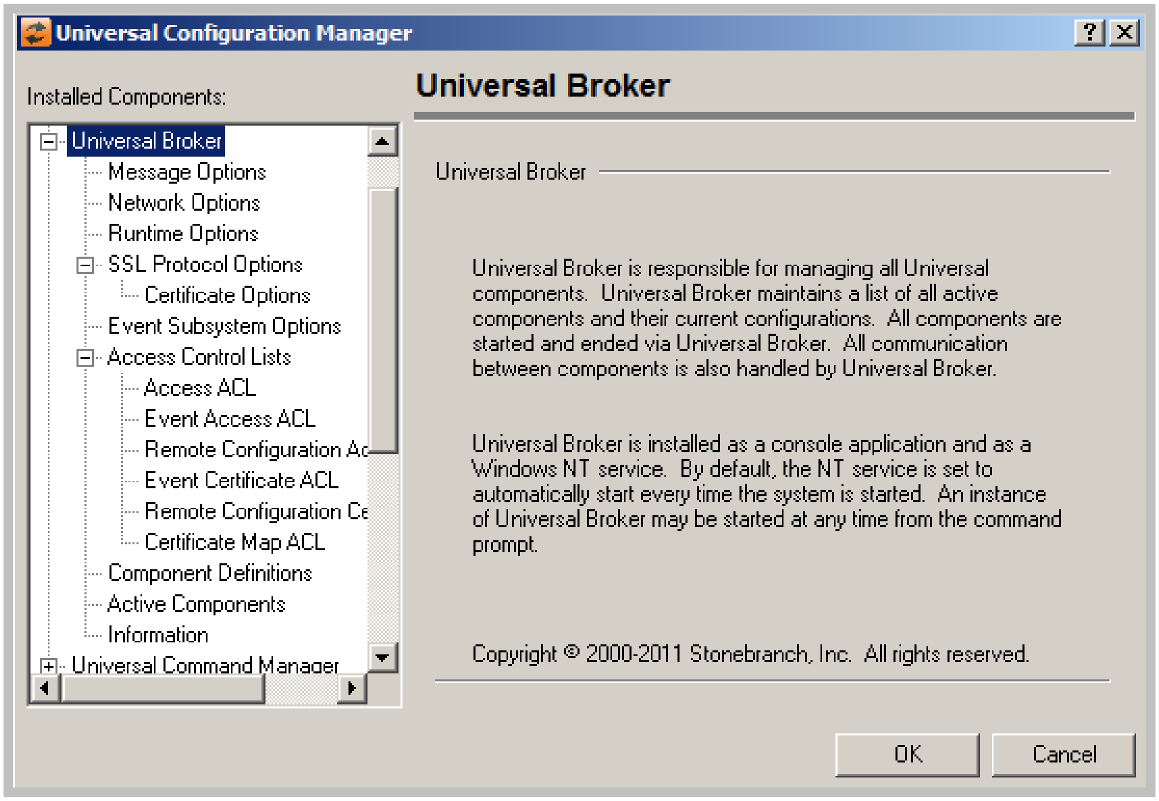
Copyright © 2000-2011 Stonebranch, Inc. All rights reserved.

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 Broker

Universal Broker

Universal Broker is responsible for managing all Universal components. Universal Broker maintains a list of all active components and their current configurations. All components are started and ended via Universal Broker. All communication between components is also handled by Universal Broker.

Universal Broker is installed as a console application and as a Windows NT service. By default, the NT service is set to automatically start every time the system is started. An instance of Universal Broker may be started at any time from the command prompt.

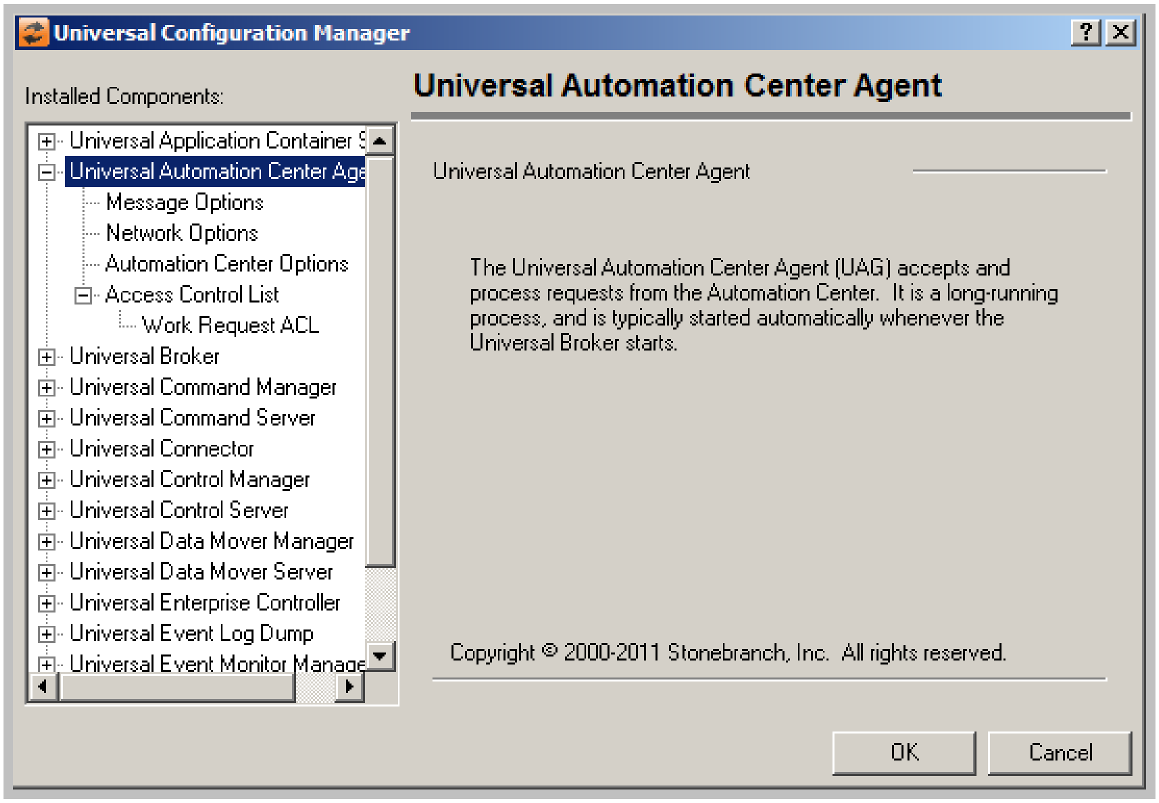
Copyright © 2000-2011 Stonebranch, Inc. All rights reserved.

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.


Universal Configuration Manager
Installed Components:
Universal Application Container S
Universal Automation Center Age
Message Options
Network Options
Automation Center Options
Access Control List
Work Request ACL
Universal Broker
Universal Command Manager
Universal Command Server
Universal Connector
Universal Control Manager
Universal Control Server
Universal Data Mover Manager
Universal Data Mover Server
Universal Enterprise Controller
Universal Event Log Dump
Universal Event Monitor Manage
Universal Automation Center Agent
Universal Automation Center Agent
OK
Cancel

The Universal Automation Center Agent (UAG) accepts and process requests from the Automation Center. It is a long-running process, and is typically started automatically whenever the Universal Broker starts.

Copyright © 2000-2011 Stonebranch, Inc. All rights reserved.

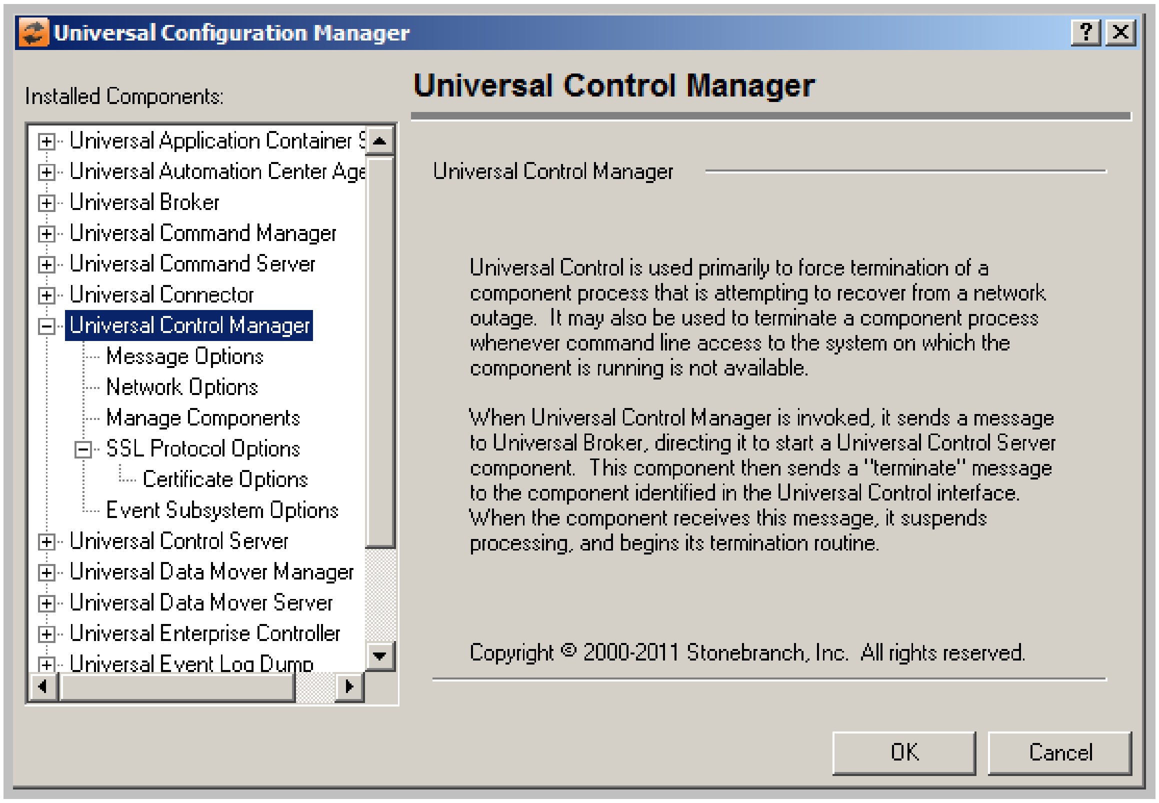
Workload Automation 5 Utilities Installed Components

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 Manager

Universal Control Manager

Universal Control is used primarily to force termination of a component process that is attempting to recover from a network outage. It may also be used to terminate a component process whenever command line access to the system on which the component is running is not available.

When Universal Control Manager is invoked, it sends a message to Universal Broker, directing it to start a Universal Control Server component. This component then sends a "terminate" message to the component identified in the Universal Control interface. When the component receives this message, it suspends processing, and begins its termination routine.

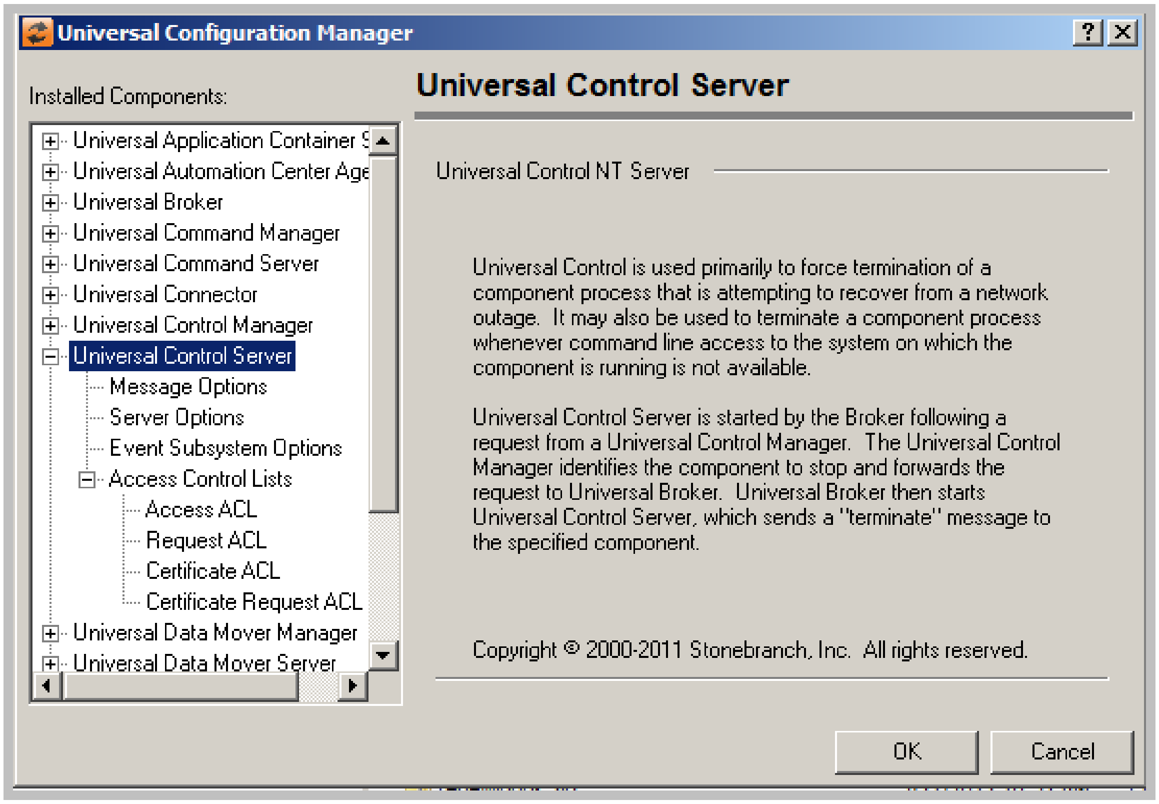
Copyright © 2000-2011 Stonebranch, Inc. All rights reserved.

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 Control Server

Universal Control NT Server

Universal Control is used primarily to force termination of a component process that is attempting to recover from a network outage. It may also be used to terminate a component process whenever command line access to the system on which the component is running is not available.

Universal Control Server is started by the Broker following a request from a Universal Control Manager. The Universal Control Manager identifies the component to stop and forwards the request to Universal Broker. Universal Broker then starts Universal Control Server, which sends a "terminate" message to the specified component.

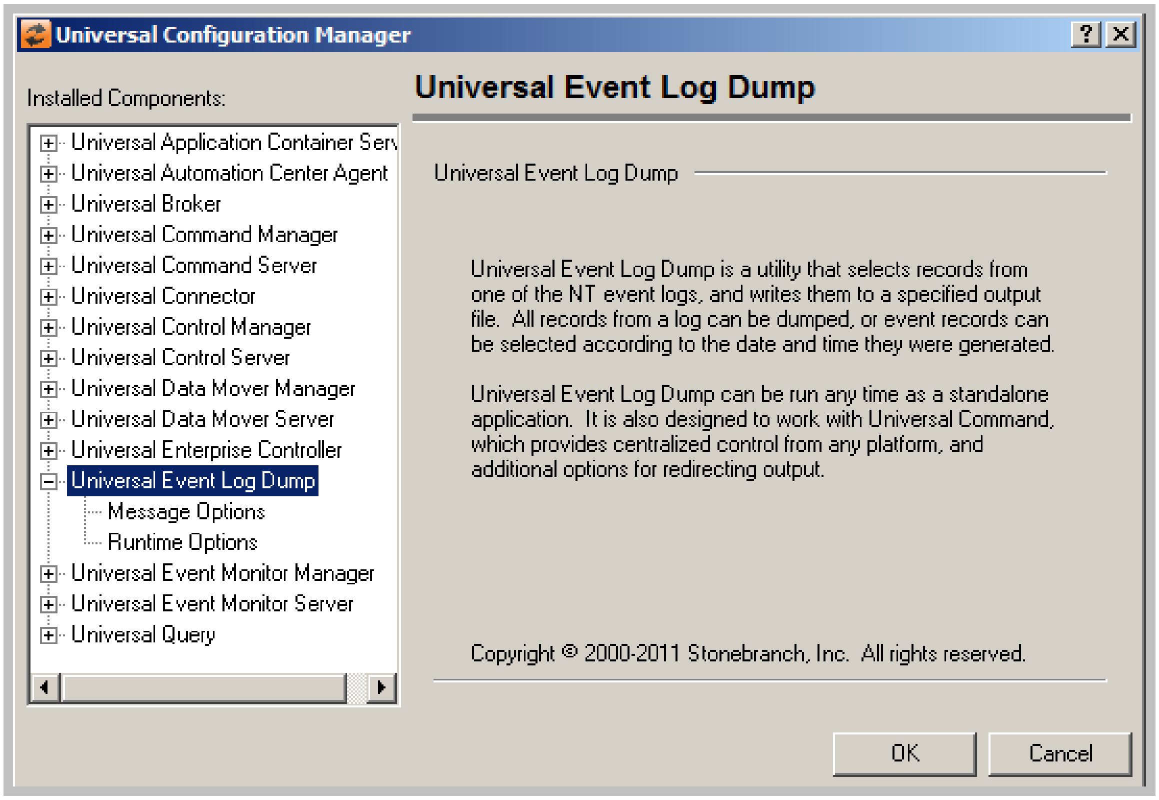
Copyright © 2000-2011 Stonebranch, Inc. All rights reserved.

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 Configuration Manager
Installed Components:
Universal Application Container Serv
Universal Automation Center Agent
Universal Broker
Universal Command Manager
Universal Command Server
Universal Connector
Universal Control Manager
Universal Control Server
Universal Data Mover Manager
Universal Data Mover Server
Universal Enterprise Controller
Universal Event Log Dump
Message Options
Runtime Options
Universal Event Monitor Manager
Universal Event Monitor Server
Universal Query
OK
Cancel

Universal Event Log Dump

Universal Event Log Dump

Universal Event Log Dump is a utility that selects records from one of the NT event logs, and writes them to a specified output file. All records from a log can be dumped, or event records can be selected according to the date and time they were generated.

Universal Event Log Dump can be run any time as a standalone application. It is also designed to work with Universal Command, which provides centralized control from any platform, and additional options for redirecting output.

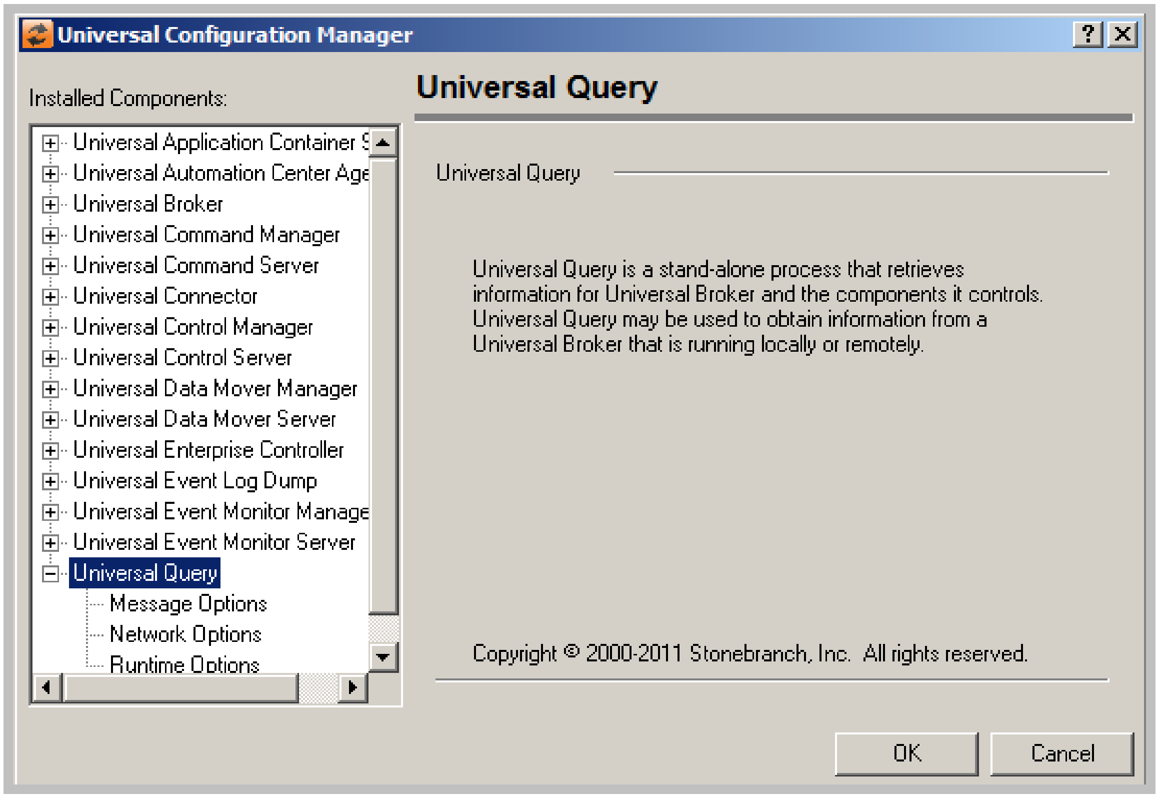
Copyright © 2000-2011 Stonebranch, Inc. All rights reserved.

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.



Universal Configuration Manager
Installed Components:
Universal Application Container S
Universal Automation Center Age
Universal Broker
Universal Command Manager
Universal Command Server
Universal Connector
Universal Control Manager
Universal Control Server
Universal Data Mover Manager
Universal Data Mover Server
Universal Enterprise Controller
Universal Event Log Dump
Universal Event Monitor Manage
Universal Event Monitor Server
Universal Query
Message Options
Network Options
Runtime Options
Universal Query
Universal Query
OK
Cancel

Universal Query is a stand-alone process that retrieves information for Universal Broker and the components it controls. Universal Query may be used to obtain information from a Universal Broker that is running locally or remotely.

Copyright © 2000-2011 Stonebranch, Inc. All rights reserved.

Configuration 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.

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:

Step 1	Read its configuration file. Universal Broker refreshes its configuration options.
Step 2	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.
Step 3	Read the Universal Access Control List configuration entries from the registry. The Broker replaces its UACL entries with the newly read entries.

Configuration Refresh for Universal Broker

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 Overview

Refreshing via Universal Control Examples

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.)

- 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
- Refreshing Universal Broker from HP NonStop
- Refreshing a Component from HP NonStop

⚠ **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 zOS

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
-refresh	Instruction to refresh Universal Broker on the remote system.
-host	Directs the command to a computer with a host name of **dallas**.

Universal Broker Actions

The refresh request directs the Broker to take the following actions:

Step 1	Read its configuration file. The Broker refreshes configuration options.
Step 2	Read all component definitions found in ddname **UNVCONF**. The Broker replaces all component definitions with the newly read component definitions. New component definitions are added and deleted component definitions are removed.

Step 3	Read the Universal Access Control List configuration file allocated to ddname **UNVACL**. 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

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 **dallas**.
-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

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 **dallas**.
-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:

Step 1	Read its configuration file. Universal Broker refreshes its configuration options.
Step 2	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.
Step 3	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

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 **"UEM-dallas"** to the started component.
-host	Directs the command to a computer with a host name of **dallas**.
-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

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 **dallas**.
-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:

Step 1	Read its configuration file **ubroker.conf**. Universal Broker refreshes the following configuration options: • MESSAGE_LANGUAGE • RUNNING_MAX
Step 2	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.
Step 3	Read the Universal Access Control List configuration file **uacl.conf**. 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

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.
-cmdid	Assigns a command identifier of **"UEM-dallas"** to the started component.
-host	Directs the command to a computer with a host name of **dallas**.
-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 IBM i

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 **dallas**.
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:

Step 1	Read its configuration file **UNVCONF** and member **UBROKER**.
Step 2	Read all component definitions found in the component definition file, **UNVPRD510** / **UNVCOMP**. Universal Broker replaces all component definitions with the newly read component definitions. New component definitions are added and deleted component definitions are removed.
Step 3	Read the Universal Access Control List configuration file **UNVCONF** and member **UACL**. 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

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 **'UEM-dallas'** to the started component.
HOST	Directs the command to a computer with a host name of **dallas**.
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 HP NonStop

Refreshing Universal Broker via Universal Control from HP NonStop

This example refreshes Universal Broker on a remote system.

```
run uctl –refresh -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.
-host	Directs the command to a computer with a host name of **dallas**.
-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:

Step 1	Read its configuration file **UBRCFG**. Universal Broker refreshes the following configuration options: • MESSAGE_LANGUAGE • RUNNING_MAX
Step 2	Read all component definitions found in the component definition subvolume. Universal Broker replaces all component definitions with the newly read component definitions. New component definitions are added and deleted component definitions are removed.
Step 3	Read the Universal Access Control List configuration file **UACLCFG**. Universal Broker replaces its UACL entries with the newly read entries.

Components

Universal Control

Refreshing a Component from HP NonStop

Refreshing a Component via Universal Control from HP NonStop

This example refreshes a component on a remote system.

```
run 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.
-cmdid	Assigns a command identifier of **"UEM-dallas"** to the started component.
-host	Directs the command to a computer with a host name of **dallas**.
-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

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

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 Workload Automation 5 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

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
installation_directory	“C:\Program Files\Universal\UCmdMgr”
message_level	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

Merge Configuration Files Introducing New Options

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
-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.
-keep_nomatch	Controls merge behavior when an option in -dest has no match in -source.

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
installation_directory	“C:\Program Files\Universal\UCmdMgr”
message_level	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

Merge Configuration Files Using Installation-Dependent Values

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
-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.
-cfgtype	Notifies UPIMERGE that -source is a configuration file that contains settings for the specified Workload Automation application.
-installdir	Primary location in which the Workload Automation application identified by -cfgtype 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
installation_directory	“D:\Program Files\Universal\UCmdMgr”
message_level	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

Indesca - Component Management

Component Management - Overview

Component Management

Component Management information for Indesca 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 Indesca Server component (for Universal Command, 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

Indesca Component Definition Options

The following component definition options are available for Indesca components:

Universal Broker Component Definition Options

Universal Automation Center Agent Component Definition Options

Universal Command Component Definition Options

UAC Server Component Definition Options

Universal Event Monitor Component Definition Options

Universal Control Component Definition Options

Starting and Stopping Components

Starting Components

There are four ways in which Indesca 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 Command 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 Universal Control START_CMD option, that do not require interaction with a Manager. Currently, only two Indesca 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 Indesca Server component can 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 Indesca 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 Broker for z/OS
- Starting Universal Broker for Windows
- Starting Universal Broker for UNIX
- Starting, Ending, Working with Universal Broker for IBM i
- Starting Universal Broker for HP NonStop

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
- Stopping an HP NonStop Component via Universal Control

⚠ **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 Broker and Universal Control, substitute the tagged names for these untagged names. (For information on tagged names, see UCHGRLS (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

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\nls**
- **.\Universal\UCmdMgr**
- **.\Universal\UCtlMgr**
- **.\Universal\UDMMgr**
- **.\Universal\UEld**
- **.\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

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 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 *running* or *stopped*. If the daemon is running, the script displays its process ID.
restart	Performs a **stop** request followed by a **start** 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, Working with Universal Broker for IBM i

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 ***JOBD**) 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 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 Universal Broker for HP NonStop

Starting Universal Broker for HP Nonstop

Universal Broker for HP NonStop runs as an Open System Services (OSS) application.

It 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 only one active instance. The PID file is a locking mechanism that prevents the execution of a second Broker. The PID file, named **UBRPID**, is created in subvolume **$SYSTEM.UNVLOG** by default. If the PID file is in the PID subvolume, it is assumed that a Universal Broker instance is executing.

Console Application

The command **ubroker** starts Universal Broker as a console application.

The following figure illustrates the Universal Broker start command.

```
ubroker [OPTIONS...]
```

Console Security

The Universal Broker runs with the same user ID as the user who starts it. The Universal Broker does not require **super.super** rights. It only requires access to its installation subvolume and files.

However, components started by Universal Broker also run with the same user ID as Universal Broker. Some components may require **super.super** rights.

(See the security documentation of the components you wish to run for details on their security requirements.)

Daemon

Universal Broker can run as a daemon process. This is the preferred method of running the Broker. A daemon startup script is provided to manage the starting and stopping of the Broker daemon. The startup script utilizes the PID file to ensure only one instance of the Broker is executing at any one time. For this reason, the startup script should be used to start and stop the Broker.

> **Note**
> The Universal Broker daemon startup script and the Universal Broker daemon program file both are installed within the **$SYSTEM.UNVBIN** subvolume. The Broker daemon startup script name is **ubrokerd** and the Broker daemon program file name is **ubrd**.

```
ubrokerd { start | stop | status | restart }
```

The following table describes the command line arguments to the Universal Broker daemon startup.

Command	Description

Start	Starts the Universal Broker daemon. Only one instance of Universal Broker can run at any given time, so if the Broker is already 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 *running* or *stopped*. If the daemon is running, the script displays its process ID.
Restart	Performs a **stop** request followed by a **start** request.

Daemon Security

When a daemon is started at system initialization, it is started as user **super.super**. The **super.super** user ID provides sufficient authority for the Broker and any component it may start.

If the daemon is started with a non-super user ID, the environment is the same as if it was started as a console application (see Console Security, above).

Starting and Stopping Universal Enterprise Controller for zOS

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 require it to be stopped and restarted by the Windows Service Control Manager.

To access the Service Control Manager:

Step 1	Click the **Control Panel** on the Windows Start menu.
Step 2	Double-click the **Administrative Tools** icon on the Control Panel window.
Step 3	Double-click the **Services** icon on the Administrative Tools window.
Step 4	On the Services window, select Universal Enterprise Controller in the list of services.
Step 5	In the Action menu, click: 1. Stop, to stop UEC for Windows. 2. Start, to start UEC for Windows.

Starting a zOS Component via Universal Control

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
-start	Name of the 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 **dallas**.
-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 a zOS Component via Universal Control

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
-stop	Component to stop.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the stop request.
-pwd	Password for the user ID.

Components

Universal Control

Starting a Windows Component via Universal Control

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
-start	Name of the 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 **dallas**.
-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 a Windows Component via Universal Control

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
-stop	Component to stop.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the stop request.
-pwd	Password for the user ID.

Components

Universal Control

Starting a UNIX Component via Universal Control

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
-start	Name of the 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 **dallas**.
-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 a UNIX Component via Universal Control

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
-stop	Component to stop.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the stop request.
-pwd	Password for the user ID.

Components

Universal Control

Starting an IBM i Component via Universal Control

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 **dallas**.
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

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 **dallas**.
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

Stopping an HP NonStop Component via Universal Control

Stopping an HP NonStop Component via Universal Control

This example stops a component on a remote system.

```
run uctl –stop 10739132 -host dallas -userid joe –pwd akkSdiq
```

Command Line Options

The command line options used in this example are:

Option	Description
-stop	ID of the component on the remote system to stop.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the command. 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

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.

-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.

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
-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.
-delete	Delete Agent definitions from UEC.
-broker_name	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.

-pwd	Password associated with -userid.
-add	Add Agent definitions to specified group(s).
-deffile	File containing multiple broker definitions to be added or deleted in the UEC database.
-groups	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
-port	TCP/IP port number of the UEC.
-userid	UEC user ID/account with which Brokers will be modified.
-pwd	Password associated with -userid.
-delete	Delete Agent definitions from specified group(s).
-deffile	File containing multiple broker definitions to be added or deleted in the UEC database.
-groups	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 - zOS

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
-export	Output the described broker definition in a format to be used by a broker definition file.
-arcfile	Archived file to retrieve for export.
-level	Level of messages written.
-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.

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
-arcfile	Archived file to retrieve for export.
-export	Output the described broker definition in a format to be used by a broker definition file.
-stime	Start time of exported data.
-etime	End time of exported data.
-level	Level of messages written.
-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.

Indesca - Messaging and Auditing

Messaging and Auditing - Overview

Indesca 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

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.)

Audit	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.
Informational	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.
Warning	Document unexpected behavior that may cause or indicate a problem.
Error	Document program errors. They provide diagnostic data to help identify the cause of the problem.
Diagnostic	Document diagnostic information for problem resolution.
Alert	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.

Audit	Specifies that all audit, informational, warning, and error messages are to be written.

Informational	Specifies that all informational, warning, and error messages are to be written.
Warning	Specifies that all warning and error messages are to be written.
Error	Specifies that all error messages are to be written.
Trace	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	Messages are written to a log file. Not all programs provide this destination. The recommended directory for log files is **`/var/opt/universal/log`**. This can be changed 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 **syslog** daemon. Not all programs provide this destination. Universal programs that execute as daemons write to the **syslog**'s daemon facility. All messages include the programs process ID. If an error occurs writing to the **syslog**, the message is written to the system console.

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	Messages are written to a log file. Not all programs provide this destination. Log files are written the **$SYSTEM.UNVLOG** subvolume. All messages written to log files include a date and time stamp and the program's process ID.

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 represent 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
-msg	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 z/OS 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

Indesca - Message Translation

Message Translation - Overview

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 z/OS
- Execute UMET from z/OS Manager (with Table on Remote Server)
- Execute UMET from z/OS Manager (with Table on z/OS)
- Execute Universal Message Translator from Windows
- Execute Universal Message Translator from UNIX
- Execute Universal Message Translator from IBM i
- Execute Universal Message Translator from HP NonStop

⚠ **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

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 zOS

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=*
//CEEDUMP  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
-table	Translation table file name.
-level	Level of messages that will be displayed.

Components

Universal Message Translator

Execute UMET from zOS Manager (with Table on Remote Server)

Execute Universal Message Translator from z/OS Manager (in a Script with Table Housed on Remote Server)

The following figure illustrates the execution of Universal Message Translator from a z/OS Universal Command Manager.

```
//S1 EXEC UCMDPRC
//SCRIPTDD DD *
/opt/universal/ucmdsrv-2.2.0/bin/umet –file /home/log.file –table\
/home/umet.table –level verbose
/*
//SYSIN    DD  *
-host dallas
-script SCRIPTDD
-userid joe
-pwd abcdefg
/*
```

Universal Message Translator is executed in order to interrogate a log file and set the return code based on the translation table.

Since the command spans two lines, the native operating system continuation character must be used:

- \ for UNIX
- ¬ for Windows

The full path to the Universal Message Translator executable must be specified for UNIX if the path is not part of the user's profile.

Script Options

The script options used in this example are:

Option	Description
-file	Input message file name. If the option is not specified, UMET reads its input from **stdin**.
-table	Translation table file name.
-level	Level of messages that will be displayed.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-host	Host name or IP address of the remote system on which to execute the script.
-script	ddname from which to read a script file. The script file is sent to the remote system by UCMD

	Manager for execution
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

Components

Universal Command Manager for z/OS

Universal Message Translator

Execute UMET from zOS Manager (with Table on zOS)

Execute Universal Message Translator from z/OS Manager (in a Script with Table Housed on z/OS)

The following figure illustrates the execution of Universal Message Translator from a z/OS Universal Command Manager.

```
//S1 EXEC UCMDPRC
//UNVIN    DD DISP=SHR,DSN=hlq.umet.table
//SCRIPTDD DD *
UCOPY > c:\temp\umet.table
umet -table c:\temp\umet.table -file c:\temp\bkup.log –level verbose
/*
//SYSIN   DD  *
-host dallas
-script SCRIPTDD
-userid joe
-pwd abcdefg
/*
```

The message table is stored and maintained on z/OS and copied down to the server upon execution. The -table option points to the table of defined return codes based on text. The -file option points to the text file to be interrogated.

The first command copies the messages table from the **UNVIN** DD of the manager process to a server file named **`c:\temp\umet.table`**. The UMET program then is executed to interrogate the log file and set the return code based on the translation table.

Script Options

The script options used in this example are:

Option	Description
-table	Translation table file name.
-file	Input message file name. If the option is not specified, UMET reads its input from **stdin**.
-level	Level of messages that will be displayed.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-host	Host name or IP address of the remote system on which to execute the script.
-script	ddname from which to read a script file. The script file is sent to the remote system by UCMD Manager for execution

-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

Components

Universal Command Manager for z/OS

Universal Message Translator

Execute Universal Message Translator from Windows

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
-table	Translation table file name.
-level	Level of messages that will be displayed.
-file	Input message file name. If the option is not specified, UMET reads its input from **stdin**.

Components

Universal Message Translator

Execute Universal Message Translator from UNIX

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
-table	Translation table file name.
-level	Level of messages that will be displayed.
-file	Input message file name. If the option is not specified, UMET reads its input from **stdin**.

Components

Universal Message Translator

Execute Universal Message Translator from IBM i

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
TBL [TBLMBR]	Translation table file name.
MSGLEVEL	Level of messages that will be displayed.
MSGFILE [MSGMBR]	Input message file name. If the option is not specified, UMET reads its input from **stdin**, which is allocated to the terminal for interactive jobs and to QINLINE for non-interactive jobs.

Components

Universal Message Translator

Execute Universal Message Translator from HP NonStop

Execute Universal Message Translator from HP NonStop

The following figure illustrates the execution of Universal Message Translator from HP NonStop.

```
run $SYSTEM.UNVBIN.umet -table umettable –file umetfile -level verbose
```

The -table 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 -file option. All messages will be written to stdout.

Command Line Options

The command line options used in this example are:

Option	Description
-table	Translation table file name.
-level	Level of messages that will be displayed.
-file	Input message file name. If the option is not specified, UMET reads its input from **stdin**.

Components

Universal Message Translator

Indesca - Monitoring and Alerting

Monitoring and Alerting - Overview

Monitoring and Alerting

The Monitoring and Alerting feature of Indesca provides for the monitoring the status and activity of all Indesca 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

Indesca provides for the continuous monitoring of all Agents in an enterprise through its Universal Enterprise Controller component.

Monitored Information

Indesca 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

Indesca 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

Indesca 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

Indesca 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
- Querying - Universal Query for HP NonStop

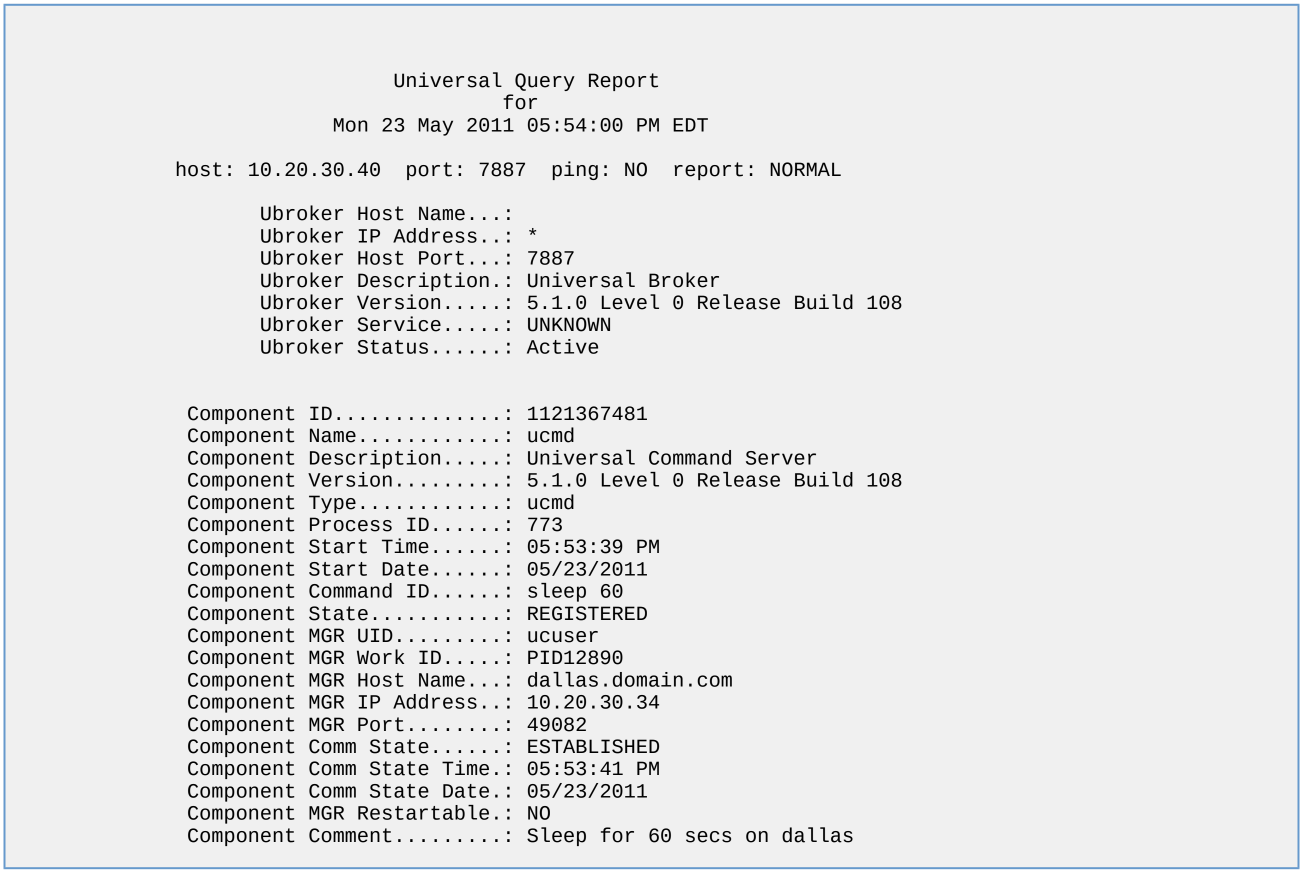
⚠ **Note**
The IBM i example references the IBM i command by its untagged name. If you are using commands with tagged names to run Universal Query, substitute the tagged name for this untagged name. (For information on tagged names, see UCHGRLS (Change Release Tag) Program.)

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 **dallas**.

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
-host	Directs the command to the **localhost**.

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 **localhost**.
PORT	TCP port on the remote server.

Components

Universal Query

Querying - Universal Query for HP NonStop

Universal Query for HP NonStop

The Universal Query utility is used to list all active components on a remote server.

The output will be written to stdout.

```
run $SYSTEM.UNVBIN.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
-host	Directs the command to the **localhost**.

Components

Universal Query

Indesca - Windows Event Log Dump

Windows Event Log Dump - Overview

Windows Event Log Dump

Indesca 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 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 z/OS Manager
- Execute Universal Event Log Dump from a Windows Server

Execute Universal Event Log Dump from zOS Manager

Execute Universal Event Log Dump from z/OS Manager

The following figure illustrates the execution of Universal Event Log Dump from a z/OS Universal Command Manager.

The application log, from the previous day at 15:00 until current time, will be dumped to the stdout of the manager process to be archived.

```
//S1 EXEC UCMDPRC
//LOGONDD  DD DISP=SHR,DSN=hlq.userid(userid)
//SCRIPTDD DD *
ueld -logtype APPLICATION -stime "*-1,15:00 PM"
//SYSIN    DD *
 -script SCRIPTDD
 -encryptedfile LOGONDD
 -host dallas
/*
```

The JCL procedure **UCMDPRC** is used to execute the **ueld** command. The command is sent to a remote system named **dallas** for execution. The **UNVOUT** DD in the **UCMDPRC** points to sysout, and is where the stdout of the remote command will be written. Additional command line options are read from the encrypted file allocated to DD **LOGONDD**.

Script Options

The script options used in this example are:

Option	Description
-logtype	Event log to be dumped.
-stime	Starting date and time.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	ddname from which to read a script file. The script file is sent to the remote system by UCMD Manager for execution
-encryptedfile	File from which to read an encrypted command options file.
-host	Host name or IP address of the remote system on which to execute the script.

Components

Universal Command Manager for zOS

Universal Event Log Dump

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
-logtype	Event log to be dumped.
-stime	Starting date and time.
-file	Complete path to the file that will be used to store the selected event log records.

Components

Universal Event Log Dump

Indesca - Fault Tolerance Implementation

Fault Tolerance Implementation - Overview

Fault Tolerance Implementation

For Indesca, fault tolerance is the capabilities of its Workload Automation 5 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, fault tolerance – for both network and component managers – is implemented in two Indesca components:

- Universal Command
- Universal Connector

Network Fault Tolerance - Universal Command

Overview

Universal Command uses the TCP/IP protocol for communications over a data network. The TCP/IP protocol is a mature, robust protocol capable of resending 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.

As with any application using TCP/IP, Universal Command is subject to these network errors. Should they occur, a product can no longer communicate and must shut down or restart. These types of errors normally show themselves as premature closes, connection resets, time-outs, or broken pipe errors.

Network Fault Tolerant Protocol

Universal Command provides the ability to circumvent these types of errors with its Network Fault Tolerant protocol. By using this protocol, Universal Command traps the connection termination caused by the network error and reestablishes the network connections. When connections have been reestablished, processing resumes automatically from the location of the last successful message exchange. No program restarts are required and no data is lost.

The Network Fault Tolerant protocol acknowledges successfully received messages and checkpoints successfully sent messages. This reduces data throughput. Consequentially, the use of network fault tolerance should be weighed carefully 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 manager enters a network reconnect phase. In this phase, the manager attempts to connect to the server and reestablish its network connections. The condition that caused the network error can persist only for seconds, or it can persist for days.

The manager attempts server reconnection for a limited amount of time, as specified by the following configuration options:

- RECONNECT_RETRY_COUNT (number of retry attempts)
- RECONNECT_RETRY_INTERVAL (frequency of retry attempts)

If all attempts fail, the manager ends with an error.

When a network connection terminates, the server's action depends on whether or not it is executing with manager fault tolerance.

Without manager fault tolerance, 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. When that time has expired, the server terminates the user process and exits.

With manager fault tolerance, the server continues executing in a disconnected state. The server satisfies all user process standard I/O requests. The user process does not block. It continues to execute normally. When the user process ends, the server waits for a manager reconnect for a period of time defined by the JOB_RETENTION option.

Universal Command Manager

Universal Command Manager starting with version 2.1.0 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 option is used to request the protocol.

Universal Command Server

Universal Command Server starting with version 2.1.0 can be configured with or without the network fault tolerant protocol. The NETWORK_FAULT_TOLERANT option is used to configure the network fault tolerant protocol.

If the server is configured with network fault tolerance off, the manager cannot override it. If the server is configured with the network fault tolerance on, the manager option determines if the protocol is actually used or not.

Manager Fault Tolerance - Universal Command

Manager Fault Tolerance - Universal Command - Overview

Manager Fault Tolerance - Universal Command

Distributed applications are comprised of many independent components running on host systems, throughout the enterprise, connected with a data network. Many of the host systems are in different physical locations, in different organization groups, and have different system management policies. It can be difficult to schedule individual host downtime when there are so many overlapping requirements.

Host systems must be shut down at scheduled intervals and, unfortunately, at unscheduled intervals. The impact of a system being down must be minimized by a distributed application.

With the Manager Fault Tolerant feature, Universal Command components can be shut down and restarted at a later time. After a manager has been started, it can be terminated and restarted at a later time. It can be shut down for any period of time: seconds, days, or even months. The server can be shut down as well. When it is restarted, any work that was completed prior to the shutdown is immediately available for a manager to retrieve.

Manager Fault Tolerance - Universal Command - Functionality

Manager Fault Tolerance Functionality

The basic functionality of manager fault tolerance is:

1. Manager requests the execution of a command on a remote system.
2. Command executes on the remote system, optionally reading and writing data.
3. Manager redirects:
   - Its standard input data to the standard input of the remote command.
   - Standard output file and standard error file of the remote command to its own standard output and standard error.

If the manager is terminated or the manager's host system is shut down, the remote command cannot read the manager’s standard input or write its standard output and error files. Without manager fault tolerance, the remote command must terminate, since its data source and destination are now gone. Otherwise, it would wait forever.

Manager fault tolerance provides an execution environment in which the manager is not required in order for the user process to continue execution on the remote system. The user process can execute to completion with or without a manager connected.

When the manager starts a user process, the manager executes as normal; standard output and standard error files are redirected back to the manager as the user process produces the data. The difference is data spooling. In order for the user process to have real-time access to its input and output, the data is spooled in the Universal Spool Database. The spool provides complete independence from the manager. The spool subsystem satisfies all data requirements for the user process via the Universal Command Server.

The manager can terminate and a new manager can restart and reconnect to the user process. If the user process has completed, the new manager receives the user processes standard files and its exit status. The restarted manager behaves in all ways as if it was the originating manager.

Command Identifier

A manager requests manager fault tolerance with the MANAGER_FAULT_TOLERANT option and by providing a command identifier (command ID) using the COMMAND_ID option. The command ID identifies the unit of work being executed. In this context, a unit of work includes the manager, server, and user process.

The manager indicates to the server that this request is restartable. The COMMAND_ID option provides a command identifier that uniquely identifies the server and user process on the remote host. When a manager is restarted, it must provide the same command ID identifying the server and user process with which it wants to reconnect.

Providing a unique command ID is not trivial. Many managers may be executing on many different hosts, and all executing work on the same server host. It is possible for a manager to start a restartable command from one host, terminate, and restart on a completely different host.

The command ID value can be any text value of unrestricted length. In practical terms, the character set and limits on command line length of the manager host impose restrictions on the value.

Standard I/O Files

The Universal Spool system satisfies all user process data requests via the Universal Command Server. When the user process reads from its standard input file, the server reads it from the spool and provides it to the user process. When the user process writes to standard output or error, the server receives the data and writes it to the spool.

A manager requesting restart capability (manager fault tolerance) first transfers its entire standard input file to the server, which it in turns writes to the spool. When all data has been received, the server creates the user process. This provides complete manager independence for the entire life of the user process.

As long as the manager is connected, the standard output and standard error files are transferred to the manager, as the user process produces the data, all in real-time. The data also is written to the spool. If the manager terminates, the data is written to the spool only.

A restarted manager is sent all of the standard output and standard error files, from the beginning, that currently is spooled. If the user process still is executing, the restarted manager will receive all of the data currently spooled. When it has caught up with the data being produced, the manager starts to receive the data from the user process as it is written.

Requesting Restart

When a restartable manager is initiated, it is either an initial instance or a restarted instance of a command ID. The command ID identifies a unit of work represented by the manager, server, and user process. See Command Identifier, above, for more information on the command ID.

The RESTART option specifies whether or not the manager instance is requesting a restart of a previous command ID. Possible RESTART values are **yes**, **no**, and **auto**.

The **auto** value specifies:

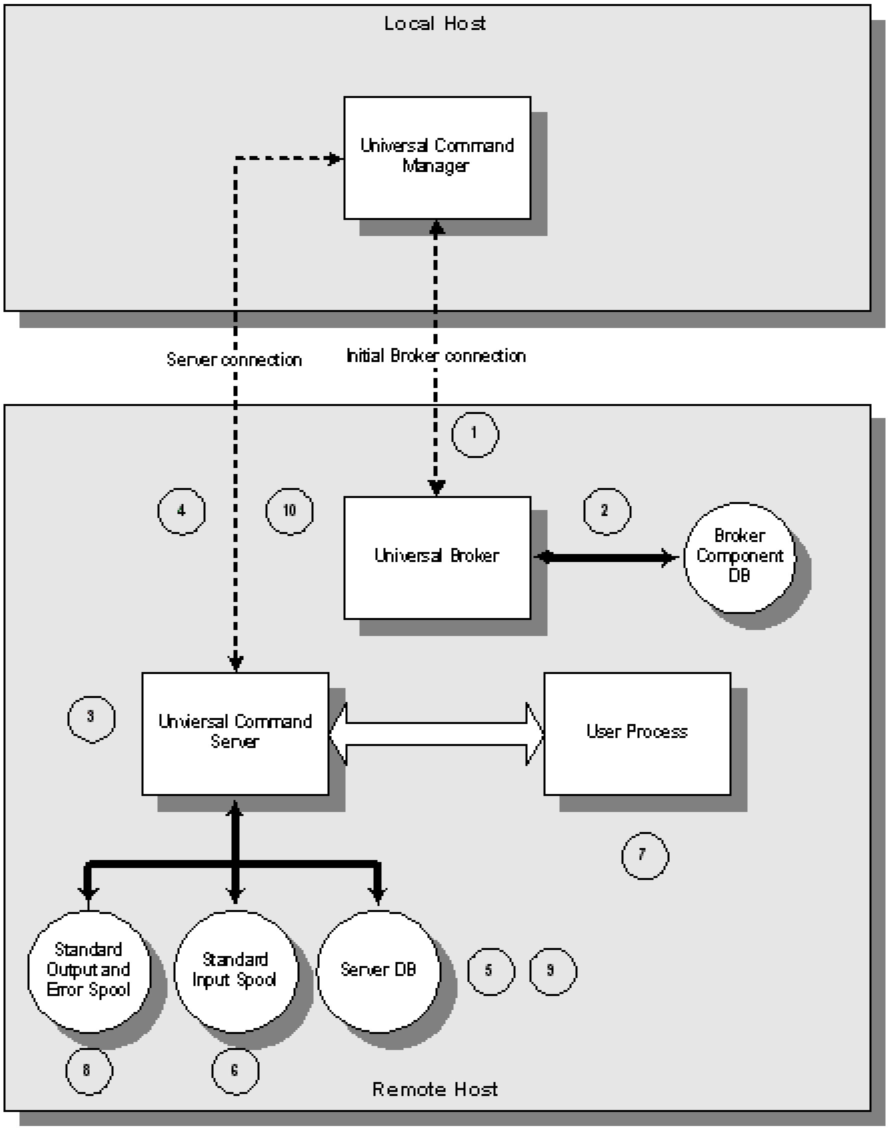
- If there is no existing command ID executing on the remote host, consider this manager execution the first instance.
- If there is an existing command ID, and it is not connected to any manager, consider this a restart of the command ID.

The **auto** value permits automatic restart by eliminating the need to modify the RESTART value for the initial instance and restarted instance.

⚠ **Note**
The **auto** value cannot be used with a COMMAND_ID value of *****, which specifies that the UCMD Manager will generate a unique command ID for each run.

Case Example 1 - Normal Execution

The following figure diagrams the sequence of events that occur when a restartable manager requests the execution of a command on a remote host. In this case the manager and server remain executing and connected until normal completion of the user process.



The Local Host is the host on which the manager is being executed.

The Remote Host is the host on which the manager is requesting command execution.

Components

The components involved are:

- Universal Command Manager
  The manager requests remote execution of a command or script. The manager executes the remote command in a manner such that the command appears to be executed locally.
- Universal Broker\
  The Broker manages Workload Automation 5 component execution.
- Universal Command Server
  The server executes the manager requested command and processes the user process's standard I/O requests.
- User Process
  The user process represents the manager requested command.

Sequence of Events

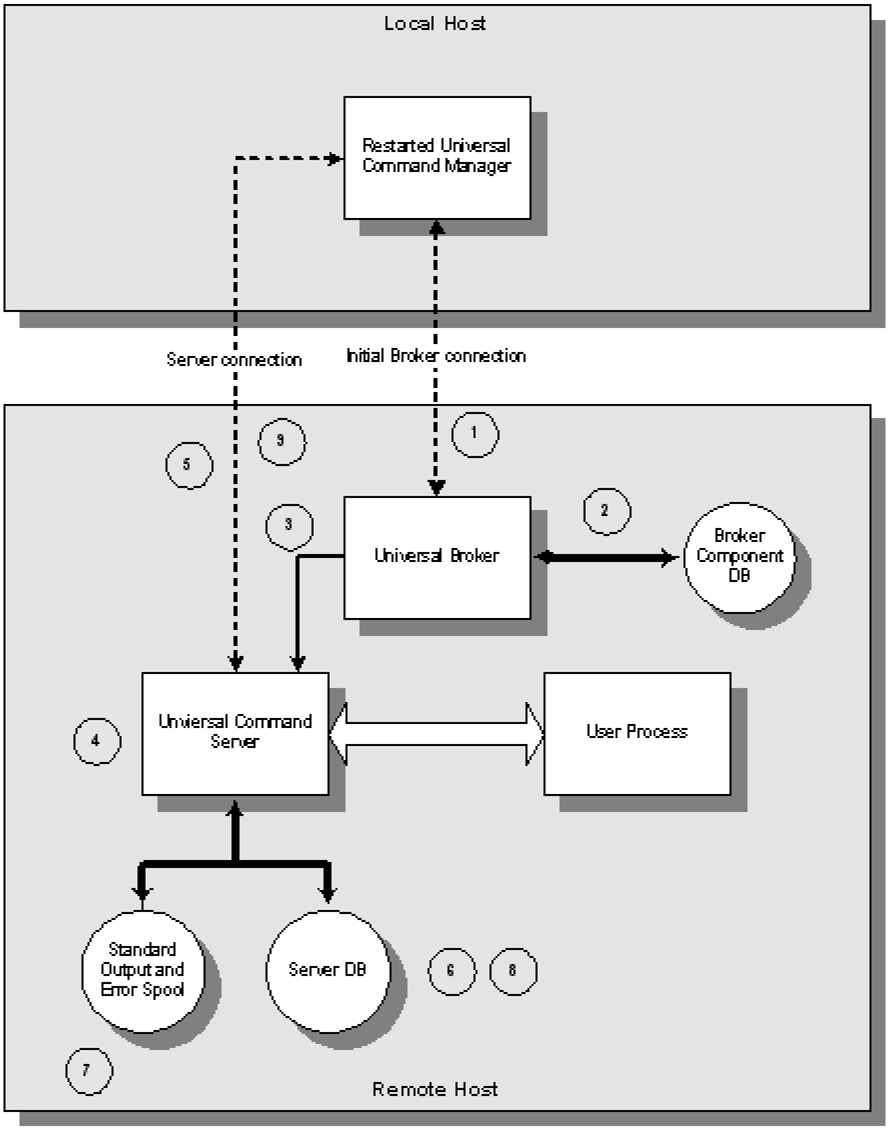
The diagram demonstrates the sequence of events that occur when a restartable manager requests command execution on a remote host. The numbers enclosed in circles represent the sequence of events and correspond to the listed descriptions below.

Step 1	The manager connects to the broker and sends a request to start a Universal Command Server. The start request from the manager requests manager fault tolerance and includes the command ID to identify the unit of work.
Step 2	The broker records the unit of work in the Broker Component Database as restartable for possible future restarts.
Step 3	The broker starts an instance of the Universal Command Server.
Step 4	The manager and server exchange messages that specify all options used to carry out the request.
Step 5	The server records the unit of work in the Universal Command Server Database for possible future restarts.
Step 6	The manager sends all standard input data to the server and the server writes the standard input data to the Universal Spool database.
Step 7	Once all standard input is spooled, the server starts the user process.
Step 8	As the user process writes standard output and standard error data, the server writes the data to the Universal Spool database. If the manager is connected to the server, the data is written to the manager as well.
Step 9	The user process executes until completion. Once the user process completes, the server writes the exit status of the user process to the Universal Command Server Database.
Step 10	The server sends the exit status to the manager. This completes the unit of work.

Case Example 2 - Restart when User Process is Executing

The following figure diagrams the sequence of events that occur when a manager requests a restart of a currently executing unit of work. In this case the initial instance of the manager terminated. A restarted instance of the manager is started and requests to be reconnected to the unit of work.

This example continues from Case Example 1 - Normal Execution. Please refer to Case Example 1 for details of the component descriptions included in the diagram.



Sequence of Events

The diagram demonstrates the sequence of events that occur when a manager requests to be restarted with a unit of work identified by a command ID. The numbers enclosed in circles represent the sequence of events and correspond to the listed descriptions below.

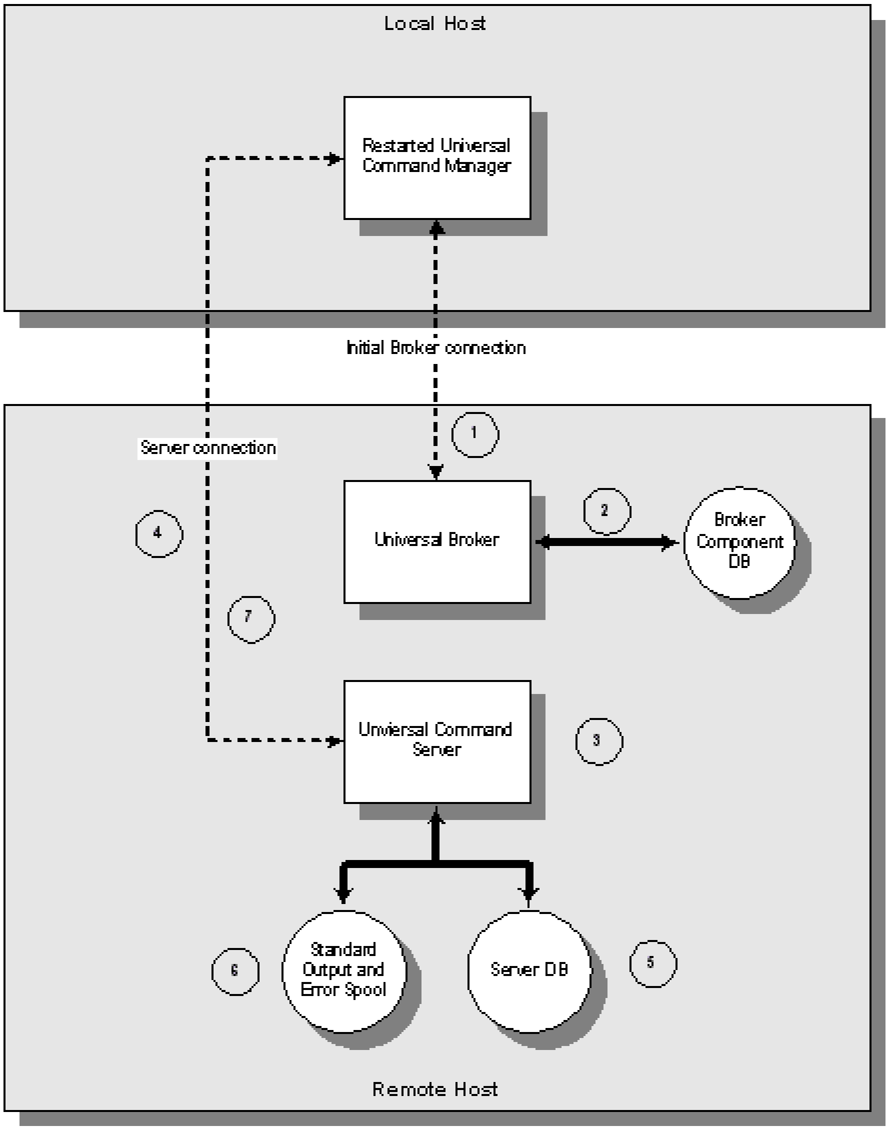
Step 1	The restarted instance of the manager sends a restart request to the broker. The restart request contains the command ID specified as part of the invocation of the manager.
Step 2	The broker verifies that the component is restartable and that the components communication state is acceptable for a restart request. If the server component were currently connected to a manager, its communication state would not permit a restart request.
Step 3	The broker sends the restart request to the server corresponding to the command ID.
Step 4	The server authenticates the request with the manager-supplied user ID and password. The password must be the same as the initial manager instance.
Step 5	The manager and server exchange options that are used to carry out the request.
Step 6	The server records the restart in the Universal Command Server Database.
Step 7	The server sends spooled standard output and error files to the manager. This is performed while the user process may still be writing standard output and error to the spool. Once all spooled output is sent to the manager, the server will send standard output and error from the user process as it is being produced.
Step 8	The user process executes to completion. The server records the user process exit status in the Universal Command Server Database.

Step 9	The server sends the exit status to the manager. This completes the unit of work.

Case Example 3 - Restart when User Process has Ended

The following figure diagrams the sequence of events that occur when a manager requests a restart of a unit of work that has completed. In this case the initial instance of the manager has terminated, the user process completed normally, and a restarted instance of the manager is started and requests to be reconnected to the completed unit of work.

This example continues from Case Example 1 - Normal Execution. Please refer to Case Example 1 for details of the component descriptions included in the diagram.



Sequence of Events

The diagram demonstrates the sequence of events that occur when a manager requests to be restarted with a unit of work identified by a command ID. The user process in this case has completed execution. The numbers enclosed in circles represent the sequence of events and correspond to the following descriptions:

Step 1	The restarted instance of the manager sends a restart request to the broker. The restart request contains the command ID specified as part of the invocation of the manager.
Step 2	The broker verifies that the component is restartable and that the components communication state is acceptable for a restart request. If the server component were currently connected to a manager, its communication state would not permit a restart request.

Step 3	Since the user process has completed, the broker starts a new server to process the restart request. The server authenticates the request with the manager-supplied user ID and password. The password must be the same as the initial manager instance.
Step 4	The manager and server exchange options that are used to carry out the request.
Step 5	The server records the restart in the Universal Command Server Database.
Step 6	The server sends spooled standard output and error files to the manager.
Step 7	The server sends the user process exit status to the manager. This completes the unit of work.

Manager Fault Tolerance - Universal Command - Component Management

Overview

In order to fully understand Universal Command 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 Command 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
STARTED	NO	NO	Server has started. If the server is restartable it is receiving the standard input file from the manager and spooling it.
ESTABLISHED	NO	NO	Server and manager are connected and processing normally. This is the most common state when all is well.
DISCONNECTED	YES	YES	Server is not connected to the manager. This occurs when a network error has occurred, the manager halted, or the manager host halted. The server is executing with either the network fault tolerant protocol, is restartable, or both. ⚠ **Note** The server cannot tell if the manager is still executing or not since it cannot communicate with it.
ORPHANED	NO	YES	Manager has terminated. The manager sends a termination message to the server to notify it of its termination prior to terminating. This state only occurs if the server is restartable.
RECONNECTING	NO	NO	Server has received a reconnect request from the manager to recover a lost network connection. This state should not remain long, only for the time it takes to re-establish the network connections.

RESTARTING	NO	NO	Server has received a restart request from the manager. This state should not remain long, only for the time it takes to re-establish network connections.
PENDING	NO	YES	A restartable server and its user process have completed. The user process standard output and error files are in the spool. A manager has not been restarted to pick up the spooled files and user process exit status. The server remains in this state until a manager is restarted.
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.

Network Fault Tolerance - Universal Connector

Overview

Universal Connector commands are processed by calling appropriate BAPI functions in the SAP system. The BAPI function calls are issued over an RFC connection. Universal Connector provides fault tolerance at the RFC level. If an RFC call fails, that call is retried until it completes successfully, or exceeds a user definable retry limit.

If an RFC call fails, Universal Connector will close the current RFC connection and establish a new RFC connection in order to continue processing. The process of establishing and preparing an RFC connection is referred to in this document as the RFC logon process. The RFC logon process involves establishing an RFC connection, logging on to the XMI interface and setting the XMI audit level. If the RFC logon process fails, it will be retried until it completes successfully, or exceeds a user definable retry limit. When the new RFC connection is successfully established, Universal Connector will reissue the failed RFC call.

The entire process of establishing a new RFC session and reissuing the failed RFC call will be retried until either

- RFC call completes successfully.
- User-definable RFC retry limit is exceeded.

Certain BAPI functions should not be retried in an unknown state. Those BAPI functions are points of failure within the Universal Connector fault tolerant solution.

Points of Failure, below, lists the points of failure and their relationship to Universal Connector commands.

Points of Failure

The points of failure within Universal Connector fault tolerant architecture are:

- Job Submission
  Some BAPI functions called in the submission process cannot be restarted in an unknown state without possible negative consequences. If an RFC call fails issuing those BAPIs, Universal Connector will end unsuccessfully.
- Job Modification
  Some BAPI functions called in the job modification process can not be restarted in an unknown state without possible negative consequences. If an RFC call fails issuing those BAPIs, Universal Connector will end unsuccessfully.
- Job Start
  Some BAPI functions called in the job start process can not be restarted in an unknown state. If an RFC call fails issuing those BAPIs, Universal Connector will end unsuccessfully.

Network Fault Tolerance Configuration Parameters

The following set of Universal Connector configuration options can be used to fine-tune the fault tolerance support for a particular environment:

- LISTEN_INTERVAL (-rfc_listen_interval)
- LOGON_RETRY_COUNT (-rfc_logon_retry_count)
- LOGON_RETRY_INTERVAL (-rfc_logon_retry_interval)
- SECURE_CFT (-rfc_retry_count)
- RETRY_CALL_INTERVAL (-rfc_retry_interval)
- TIMEOUT_INTERVAL (-rfc_timeout)

See RFC (Remote Function Call) Options for details concerning the use of these parameters.

Client Fault Tolerance - Universal Connector

Client Fault Tolerance - Universal Connector - Overview

Client Fault Tolerance - Universal Connector

The Client Fault Tolerance feature allows the Universal Connector client application to be shut down and restarted at a later time.

This functionality helps avoid problems that can result if the Universal Connector application terminates unexpectedly while processing an SAP job. In such an instance, the Client Fault Tolerance restart capability allows Universal Connector to reconnect to a running (or completed) job while preventing the unintentional start of a new instance of the original SAP job.

To achieve Client Fault Tolerance, Universal Connector must be able to associate the SAP jobs it defines and starts with a particular unit of work. In this context, a unit of work includes the Universal Connector client and SAP job instance.

To associate an SAP job with a particular unit of work, the user must be able to specify some identifying characteristic that is specific to that unit of work. The SAP system uniquely identifies job instances by a job name/job ID pair. Since the job name must be reusable and the job ID is assigned by the SAP system at the time of definition, Universal Connector must use an alternative job characteristic. This alternative job characteristic is the Universal Connector command Identifier.

Universal Connector references a particular unit of work by a job name/Command Identifier combination. The Universal Connector command identifier is tied to the SAP job by appending a Command ID Job Step to the SAP job associated with the Universal Connector command instance. The Command ID job step is required for identification purposes only. Therefore, the program used for the Command ID step is intended to add minimum overhead to the job. The Command ID used for the job is included in the definition of the Command ID job step.

Client Fault Tolerance - Universal Connector - Modes

Overview

Universal Connector supports two modes of client fault tolerance:

1. Secure Client Fault Tolerance (Secure CFT)
2. Pre-XBP 2.0 Client Fault Tolerance (CFT)

The mode of client fault tolerance to be used by Universal Connector is determined by the value of the SECURE_CFT option.

Valid values for this option are **yes** and **no**:

- **yes** will cause Universal Connector to use the Secure CFT mode.
- **no** will cause Universal Connector to use the original Pre-XBP 2.0 CFT mode.

The default value is **yes**.

Both modes of CFT follow the same basic process flow. When Universal Connector is requested to restart a particular command ID job, it queries the SAP system for all jobs with the specified job name. The list of jobs returned by the SAP system is scanned for a job that contains an appropriate Command ID Job Step. If found, Universal Connector will re-connect to the SAP job instance and satisfy the command line requirements.

Universal Connector is capable of restarting a command ID as long as the associated command ID job remains in the SAP system.

Secure Client Fault Tolerance (Secure CFT) Mode

This mode is an enhancement of the original implementation. The secure CFT mode requires XBP 2.0 to be installed on the SAP side of the Universal Connector connection. In this mode, an ABAP program step is used for the command ID job step.

Using an ABAP program step as the Command ID job step eliminates the security and ease of use drawbacks mentioned above for external program job steps.

- Security
  The execution of ABAP programs and the resources required by them are secured by SAP authorization checks.
- Ease of Use
  ABAP program job steps do not require a target host. They run on whichever application server the job runs on. Therefore, there are no target specific parameters required for secure CFT mode.

Pre-XBP 2.0 Client Fault Tolerance (CFT) Mode

This mode is the original implementation of client fault tolerance used prior to the release of XBP 2.0. Due to limitations in the XBP 1.0 interface, Universal Connector client fault tolerance on pre-XBP 2.0 SAP systems uses an external program step as the command ID job step.

Using an external program step as the command ID job step has the following security and ease of use drawbacks:

- Security drawback
  Using an external program job step requires the SAP user ID to have authority to run external programs. This authority cannot be given lightly for the following reason: When running an external job step, the SAP system first performs an authorization check to see if the user ID has the right to run an external program. If so, the external program is run under the user ID of the user who started the SAP system
- Ease of use drawback
  Using an external program job step requires a target host be specified for the external program to run on. This requires information about the SAP landscape that may not be readily available. Also, this presents the possibility of the Universal Connector job's parameters becoming out of sync with the SAP landscape.

Client Fault Tolerance - Universal Connector - Parameters

Client Fault Tolerance Target Host

The client fault tolerance target host parameter is only required for pre-XBP 2.0 client fault tolerance mode. If the secure CFT mode is being used, the client fault tolerance target host parameter is ignored.

As part of an external program command ID job step definition, SAP requires a target host on which to run the external program (echo). Universal Connector provides the client fault tolerance target host parameter to specify the target host for the command ID job step.

The Client Fault Tolerance Target Host is specified with the CFT_TARGET_HOST option.

Client Fault Tolerance Command Prefix

The client fault tolerance command prefix parameter is only required for pre-XBP 2.0 client fault tolerance mode. If the secure CFT mode is being used, the client fault tolerance command prefix parameter is ignored.

The external program command ID job step has the potential to be run on any operating system reachable by the SAP system. The operating system that the Command ID Job Step runs on is that which exists on the host system specified by the Client Fault Tolerance Target Host parameter. Different operating systems may require commands to be called in different ways. Therefore, the Client Fault Tolerance Command Prefix parameter allows the user to specify the prefix necessary to run the echo command on the host system specified by the Client Fault Tolerance Target Host parameter.

For example, to run the echo command on a Windows system, the following command line entry would be required for an SAP external job step: **cmd /C echo**. Therefore, the Client Fault Tolerance Command Prefix for this system would be: **cmd /C**.

The Client Fault Tolerance Command Prefix parameter is specified with the CFT_COMMAND_PREFIX option.

Secure Client Fault Tolerance Option

The mode of client fault tolerance to be used by Universal Connector is determined by the value of the SECURE_CFT option.

Valid values for this option are **yes** and **no**:

- **yes** will cause Universal Connector to use the Secure CFT mode.
- **no** will cause Universal Connector to use the original Pre-XBP 2.0 CFT mode.

The default value is **yes**.

Client Fault Tolerance ABAP Program

The client fault tolerance ABAP program parameter is only required for secure CFT mode. If the pre-XBP 2.0 CFT mode is being used, the client fault tolerance ABAP program parameter is ignored.

The client fault tolerance ABAP program parameter is used to specify the ABAP program to use for the command ID job step. Any ABAP program can be specified. The Universal Connector internal default ABAP program to use is BTCTEST. BTCTEST is a standard SAP ABAP program that should be available on all SAP systems. It does no real processing and can be considered a dummy program that does not interfere with job processing and places little overhead on the system.

Client Fault Tolerance - Universal Connector - Command ID Job Step

Overview

USAP creates Command ID jobs by appending a job step to the user's SAP job being defined to the system. This appended job step is the Universal Connector Command ID Job Step.

Pre-XBP 2.0 CFT Mode

In pre-XBP 2.0 CFT mode, the Universal Connector Command ID Job Step executes the external program echo. A string containing the command ID is inserted in the parameter field of the job step. The echo command is lightweight, does not interfere with the original job, and results in the command ID being printed to the joblog.

Secure CFT Mode

In secure CFT mode, the Universal Connector command ID job step executes an ABAP program. The ABAP program defined to the command id step is user configurable with the covetable parameter. Any ABAP program can be specified. The Universal Connector internal default ABAP program to use is BTCTEST. BTCTEST is a standard SAP ABAP program that should be available on all SAP systems. It does no real processing and can be considered a dummy program that does not interfere with job processing and places little overhead on the system.

Client Fault Tolerance - Universal Connector - Command Identifier

Command Identifier

Universal Connector requests client fault tolerance by providing a command identifier. The command identifier is specified on the command line with parameter **-cmdid**. The command ID/job name pair identifies the unit of work being executed.

The command ID option provides a command identifier that (paired with job name) uniquely identifies the SAP job on the SAP system. When a Universal Connector job is restarted, it must provide the same command ID identifying the SAP job with which it wants to reconnect.

Providing a unique command ID is not trivial. Many Universal Connector clients may be executing on many different hosts, all executing work on the same SAP system. It is possible for a Universal Connector client to start a restartable job from one host, terminate, and restart on a completely different host.

The command ID value can be any text value up to 245 characters in length. In practical terms, the character set and limits on command line length of the Universal Connector host may impose further restrictions on the value.

Client Fault Tolerance - Universal Connector - Requesting Restart

Requesting Restart

When a restartable Universal Connector command is initiated, it is either an initial instance or a restarted instance of a command ID.

The RESTART option is specified on the command line with parameter **-restart**. RESTART specifies whether or not the Universal Connector command instance is requesting a restart of a previous command ID. Possible RESTART values are **yes**, **no**, or **auto**.

The **auto** value specifies that if there is no existing command ID job on the SAP system, consider this Universal Connector execution the first instance. If there is an existing command ID job, consider this a restart of the command ID. The **auto** value permits automatic restart by eliminating the need to modify the RESTART value for the initial instance and restarted instance.

It is important to note that when using the RESTART **auto** value, Universal Connector will not start a new instance of a job on the SAP system if a job matching the job name/command ID exists in the SAP system. Universal Connector will continue to reconnect to the existing SAP job.

Without considering the behavior resulting from the use of RESTART **auto**, it may be possible for one to assume that a job has been run multiple times when, in fact, Universal Connector has been reconnecting to the same job instance. Informational messages are printed by Universal Connector to standard error to indicate the reconnected status but, if the message level is not set to **info**, the messages will not be seen.

Controlling Auto Restart

Misunderstanding the auto restart behavior described above can potentially have serious consequences. For this reason, the ALLOW_AUTO_RESTART allows or disallows the use of auto restart. You can specify ALLOW_AUTO_RESTART in the configuration file level and override it at the command line level.

ALLOW_AUTO_RESTART in Configuration File

The ALLOW_AUTO_RESTART configuration file keyword is **auto_restart_ok**.

Valid values for **auto_restart_ok** are:

- **yes** - Allows the use of auto restart.
- **no** - Prevents the use of auto restart.

ALLOW_AUTO_RESTART on Command Line

The ALLOW_AUTO_RESTART command line parameter **-autorestartok** can be used to override the configuration file setting.

Valid values for **-autorestartok** are:

- **yes** - Allows the use of auto restart
- **no** - Prevents the use of auto restart.

Sample Command Lines For Working With Client Fault Tolerance

Sample Command Lines For Working With Client Fault Tolerance

- Working With Job Definition Files
- Working With Pre-defined SAP Jobs

Working With Job Definition Files

Initial Run of a Command ID Job

The following examples will submit, start, and wait for the command ID job defined in job definition file **jobdef**. Because the RESTART option is set to **no**, Universal Connector will scan the SAP system to ensure that a command ID job with the same job name/command ID pair does not already exist on the system.

If a matching command ID job is found on the SAP system, Universal Connector will exit with an error code before performing the job submission.

Pre-XBP 2.0 CFT Mode

```
usap -userid bob -pwd secret -submit jobdef -start -wait -cft_secure_cft no
     -cft_target_host pwdf0196 -cft_cmd_prefix "cmd /C" -cmdid E8E8E80001 -restart no
```

Note
The Client Fault Tolerance Command ID Prefix (-cft_cmd_prefix) is set up for a Windows host. In many user environments, the Client Fault Tolerance Command ID Prefix option can be specified in the configuration file and will never need to be specified on the command line. The same is true for the Client Fault Tolerance Target Host option (-cft_target_host). The secure CFT option (-cft_secure_cft) also would be set in the configuration file in most cases.

Secure CFT Mode

```
usap -userid bob -pwd secret -submit jobdef -start -wait -cft_secure_cft yes
     -cft_abap BTCTEST -cmdid E8E8E80001 -restart no
```

Note
In secure CFT mode, the (-cft_secure_cft) and (-cft_abap) parameters would most likely be specified in the Universal Connector configuration file.

Command Line Options

Command line options used in these examples are:

Command Options	Description
-userid	Remote user ID with which to execute the command.

-pwd	Password for the user ID.
-submit	Defines a job to the SAP system.
-start	Starts the newly defined job.
-wait	Wait for the SAP job to complete processing.
-cft_secure_cft	Mode of client fault tolerance to be used for the command invocation.
-cft_target_host	In pre-XBP 2.0 CFT mode, the target host to use for the command ID job step when the command ID option is used.
-cft_cmd_prefix	In pre-XBP 2.0 CFT mode, the prefix command required for the operating system of the target host.
-cft_abap	ABAP program to use for the command ID job step.
-cmdid	Identifier used to identify the unit of work represented by a USAP command and the associated SAP job.
-restart	Specification for whether or not this execution of Universal Connector is a restart of a previous client fault tolerant Universal Connector command.

Restart of a Command ID Job

In the following examples, Universal Connector is requested to restart command ID job E8E8E80001. Universal Connector will first parse the **jobdef** file to determine the jobname, and then scan the SAP system for a matching command ID job.

If a matching command ID job is found, Universal Connector reconnects to that job and satisfies the command line requirements. In this case, that means that if the job has not yet been started, it will be started, Universal Connector will wait for the job to complete (if it has not already), and the output will be returned.

If no matching command ID job is found, Universal Connector will terminate with an error code. Appropriate informational messages will be printed to standard error.

Pre-XBP 2.0 CFT Mode

```
usap -userid bob -pwd secret -submit jobdef -start -wait -cmdid E8E8E80001 -restart yes
     -cft_secure_cft no
```

Command Line Options

Command line options used in these examples are:

Command Options	Description
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

-submit	Defines a job to the SAP system.
-start	Starts the newly defined job.
-wait	Wait for the SAP job to complete processing.
-cmdid	Identifier used to identify the unit of work represented by a USAP command and the associated SAP job.
-restart	Specification for whether or not this execution of Universal Connector is a restart of a previous client fault tolerant Universal Connector command.
-cft_secure_cft	Mode of client fault tolerance to be used for the command invocation.

Secure CFT Mode

```
usap -userid bob -pwd secret -submit jobdef -start -wait -cmdid E8E8E80001 -restart yes
     -cft_secure_cft yes
```

Run a Command ID Job Using Restart AUTO

In the following examples, Universal Connector will first scan the SAP system to determine if a matching command ID job exists.

If no matching command ID job is found, Universal Connector considers this to be the initial instance of this command ID job and defines the new command ID job to the SAP system. If a matching command ID job is found, Universal Connector reconnects with the existing SAP job.

After determining if the command ID job is initial or a restart, Universal Connector satisfies the command line requirements.

Pre-XBP 2.0 CFT Mode

```
usap -userid bob -pwd secret -submit jobdef -start -wait -cft_target_host pwdf0196
     -cft_cmd_prefix "cmd /C" -cmdid E8E8E80001 -restart auto -cft_secure_cft no
```

Secure CFT Mode

```
usap -userid bob -pwd secret -submit jobdef -start -wait -cmdid E8E8E80001 -restart auto
     -cft_secure_cft yes
```

Command Line Options

Command line options used in these examples are:

Command Options	Description
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

-submit	Defines a job to the SAP system.
-start	Starts the newly defined job.
-wait	Wait for the SAP job to complete processing.
-cft_secure_cft	Mode of client fault tolerance to be used for the command invocation.
-cft_target_host	In pre-XBP 2.0 CFT mode, the target host to use for the command ID job step when the command ID option is used.
-cft_cmd_prefix	In pre-XBP 2.0 CFT mode, the prefix command required for the operating system of the target host.
-cmdid	Identifier used to identify the unit of work represented by a USAP command and the associated SAP job.
-restart	Specification for whether or not this execution of Universal Connector is a restart of a previous client fault tolerant Universal Connector command.

Working With Pre-defined SAP Jobs

Initial Run of a Command ID Job

The following examples will submit, start, and wait for the command ID job defined in the pre-existing SAP job with job name 'JOB_A' and job ID 19561301. Because the RESTART option is set to **no**, Universal Connector will scan the SAP system to ensure that a command ID job with the same job name/command ID pair does not already exist on the system.

If a matching command ID job is found on the SAP system, Universal Connector will exit with an error code before performing the job submission.

Note that the Client Fault Tolerance Command ID Prefix is set up for a Windows host. In many user environments, the Client Fault Tolerance Command ID Prefix parameter can be specified in the configuration file and will never need to be specified on the command line. The same may be true for the Client Fault Tolerance Target Host parameter.

Pre-XBP 2.0 CFT Mode

```
usap -userid bob -pwd secret -submit -jobname JOB_A -jobid 19561301 -start -wait
     -cft_target_host pwdf0196 -cft_cmd_prefix "cmd /C" -cmdid E8E8E80001 -restart no
     -cft_secure_cft no
```

Note
The Client Fault Tolerance Command ID Prefix (-cft_cmd_prefix) is set up for a Windows host. In many user environments, the Client Fault Tolerance Command ID Prefix option can be specified in the configuration file and will never need to be specified on the command line. The same is true for the Client Fault Tolerance Target Host option (-cft_target_host).

Secure CFT Mode

```
usap -userid bob -pwd secret -submit -jobname JOB_A -jobid 19561301 -start -wait
     -cft_secure_cft yes -cft_abap BTCTEST -cmdid E8E8E80001 -restart no
```

Note
In secure CFT mode, the (-cft_secure_cft) and (-cft_abap) parameters would most likely be specified in the Universal Connector configuration file.

Command Line Options

Command line options used in these examples are:

Command Options	Description

-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-submit	Defines a job to the SAP system.
-jobname	Name of the SAP job.
-jobid	Job ID of the SAP job.
-start	Starts the newly defined job.
-wait	Wait for the SAP job to complete processing.
-cft_target_host	In pre-XBP 2.0 CFT mode, the target host to use for the command ID job step when the command ID option is used.
-cft_cmd_prefix	In pre-XBP 2.0 CFT mode, the prefix command required for the operating system of the target host.
-cmdid	Identifier used to identify the unit of work represented by a USAP command and the associated SAP job.
-restart	Specification for whether or not this execution of Universal Connector is a restart of a previous client fault tolerant Universal Connector command.
-cft_secure_cft	Mode of client fault tolerance to be used for the command invocation.

Restart of a Command ID Job

In the following example, Universal Connector is requested to restart command ID job E8E8E80001. Universal Connector will scan the SAP system for a matching command ID job.

If a matching command ID job is found, Universal Connector reconnects to that job and satisfies the command line requirements. In this case, that means that if the job has not yet been started, it will be started, Universal Connector will wait for the job to complete (if it hasn't already), and the output will be returned.

If no matching command ID job is found, Universal Connector will terminate with an error code. Appropriate informational messages will be printed to standard error.

Pre-XBP 2.0 CFT Mode

```
usap -userid bob -pwd secret -submit -jobname JOB_A -jobid 19561301 -start -wait
     -cmdid E8E8E80001 -restart yes -cft_secure_cft no
```

Secure CFT Mode

```
usap -userid bob -pwd secret -submit -jobname JOB_A -jobid 19561301 -start -wait
     -cmdid E8E8E80001 -restart yes -cft_secure_cft yes
```

Command Line Options

Command line options used in these examples are:

Command Options	Description
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-submit	Defines a job to the SAP system.
-jobname	Name of the SAP job.
-jobid	Job ID of the SAP job.
-start	Starts the newly defined job.
-wait	Wait for the SAP job to complete processing.
-cmdid	Identifier used to identify the unit of work represented by a USAP command and the associated SAP job.
-restart	Specification for whether or not this execution of Universal Connector is a restart of a previous client fault tolerant Universal Connector command.
-cft_secure_cft	Mode of client fault tolerance to be used for the command invocation.

Run a Command ID Job Using Restart AUTO

In the following examples, Universal Connector will first scan the SAP system to determine if a matching command ID job exists.

If no matching command ID job is found, Universal Connector considers this to be the initial instance of this command ID job and defines the new command ID job to the SAP system. If a matching command ID job is found, Universal Connector reconnects with the existing SAP job.

After determining if the command ID job is initial or a restart, Universal Connector satisfies the command line requirements.

Pre-XBP 2.0 CFT Mode

```
usap -userid bob -pwd secret -submit -jobname JOB_A -jobid 19561301 -start -wait
     -cft_target_host pwdf0196 -cft_cmd_prefix "cmd /C" -cmdid E8E8E80001 -restart auto
     -cft_secure_cft no
```

Secure CFT Mode

```
usap -userid bob -pwd secret -submit -jobname JOB_A -jobid 19561301 -start -wait
     -cmdid E8E8E80001 -restart auto -cft_secure_cft yes
```

Command Line Options

Command line options used in these examples are:

Command Options	Description
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-submit	Defines a job to the SAP system.
-jobname	Name of the SAP job.
-jobid	Job ID of the SAP job.
-start	Starts the newly defined job.
-wait	Wait for the SAP job to complete processing.
-cft_target_host	In pre-XBP 2.0 CFT mode, the target host to use for the command ID job step when the command ID option is used.
-cft_cmd_prefix	In pre-XBP 2.0 CFT mode, the prefix command required for the operating system of the target host.
-cmdid	Identifier used to identify the unit of work represented by a USAP command and the associated SAP job.
-restart	Specification for whether or not this execution of Universal Connector is a restart of a previous client fault tolerant Universal Connector command.
-cft_secure_cft	Mode of client fault tolerance to be used for the command invocation.

Implementing Fault Tolerance - Examples

Implementing Fault Tolerance Examples

- Implementing Manager Fault Tolerance for Windows

Implementing Manager Fault Tolerance for Windows

Implementing Manager Fault Tolerance for Windows

The following figure illustrates how to activate manager fault tolerance. A unique command id is always required for manager fault tolerance.

```
ucmd -script script.file -host dallas –encryptedfile encrypted.file
     –managerft yes -cmdid uniquejobname –restart auto
```

The command is sent to a remote system, **dallas**, for execution. The output of the script is redirected back to the UCMD process. Additional command line options are read from the encrypted file, **`encrypted.file`**. Manager fault tolerance is turned on. A unique command ID, **`uniquejobname`**, is coded to identify the process.

Restart is detected automatically. If an executing or pending command ID exists, a reconnect is performed. If not, the process is started as if new.

Command Line Options

The command line options used in this example are:

Options	Description
-script	File from which to read a script file. The script file is sent to the remote system for execution.
-host	Directs the command to a computer with a host name of **dallas**.
-encryptedfile	File from which to read an encrypted command options file.
-managerft	Specification for whether or not the manager fault tolerant feature is used.
-cmdid	Unique command ID associated the unit of work.
-restart	Specification for whether or not the manager is requesting restart.

Components

Universal Command Manager for Windows

Indesca - Network Data Transmission

Network Data Transmission - Overview

Indesca - Network Data Transmission

Distributed systems, such as Workload Automation 5, communicate over data networks. All Indesca components communicate using the TCP/IP protocol. The UDP protocol is not used for any product data communication over a network.

Network Protocols

Workload Automation can utilize either of two network protocols:

1. 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 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 5 (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, the Universal Application Protocol is discussed as well.

SSL (Secure Socket Layer) Protocol

Overview

Indesca 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 version3. 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 insures 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 Universal V2 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

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 Indesca:

1. Control data (or messages) consists of messages generated by Indesca 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-NULL 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 loosing 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

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 Indesca 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 then 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.

Indesca - zOS Cancel Command Support

zOS CANCEL Command Support - Overview

z/OS CANCEL Command Support

Indesca provides network fault tolerance and, in some cases, manager fault tolerance (see Fault Tolerance Implementation). These features provide users with the ability to execute jobs that will continue to run when the network is down and when a manager is terminated.

However, there are scenarios in which the user may want to cancel an executing job that supports manager and/or network fault tolerance and have 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 Command

Overview

A user may want to cancel an executing Universal Command job that supports manager and/or 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.

When a Universal Command 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 Command Server process associated with the stopped/ended manager process.

Exit Codes

Through the use of the SERVER_STOP_CONDITIONS configuration option, the Universal Command 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 Command 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 Command 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 either network fault tolerance or manager fault tolerance is enabled, the Universal Broker will execute a **uctl** command to STOP the running Universal Command Server component.

Security Token

For security purposes, Workload Automation 5 passes around a security token that is used by the locally running Universal Broker to STOP associated Universal Command Server process.

This security token is generated on a component-by-component basis by the Universal Broker process that starts the Universal Command Server. Upon generation, this token is returned to the Universal Command 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 Command 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 Command Server process. When the token is authenticated, the Universal Command Server process is STOPPED.

zOS CANCEL Command Support - Universal Connector

Overview

A user may want to cancel an executing Universal Connector job that supports client and / or network fault tolerance and have both the Universal Connector and SAP processes terminate immediately. Because of the separation of work between Universal Connector and SAP, when the Universal Connector client is terminated, the SAP job continues to execute.

In particular, z/OS supports a CANCEL command that will terminate a job executing on the z/OS operating system. When a Universal Connector job is cancelled via the z/OS CANCEL command, the job terminates with either of these exit codes:

- S122, if job is cancelled with a dump.
- S222, if job 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 invoke a new instance of Universal Connector and issue a CANCEL command to terminate the associated SAP job.

Exit Codes

Through the use of the SERVER_STOP_CONDITIONS configuration option, the Universal Connector process notifies the locally running Universal Broker of the exit codes that should cause it to terminate the running SAP job. With this option, you can specify a list of exit codes that should trigger the locally running Universal Broker to invoke a Universal Connector process to terminate the SAP job.

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 stores this and other component-specific data about the executing manager component. When this executing Universal Connector 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, the Universal Broker will invoke a new instance of the Universal Connector to execute a CANCEL command to terminate the running SAP job.

Indesca - Glossary

Glossary

This glossary defines terms used within the Indesca business solution:

Agent

A single Workload Automation installation comprised of Universal Broker, Universal Automation Center Agent, and one or more other Workload Automation 5 components, such as Universal Command.)

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 Stonebranch workload automation 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

Indesca 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

Indesca 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.

Web 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 Indesca.

XD Connector

XD Connector supports workload execution within the WebSphere Extended Deployment environment using synchronous communication via the SOAP protocol.