Opswise Managed File Transfer 5.2.0

User Guide

Opswise Managed File Transfer 5.2.0 User Guide

Currently, IBM i runs Workload Automation 5.1.0, and HP NonStop runs Universal Command 2.1.1.

Information in this 5.2.0 User Guide for IBM i and HP NonStop refer to these versions.

Opswise Managed File Transfer Documentation

The Opswise Managed File Transfer 5.2.0 User Guide provides information on the intelligent file transfer features of Opswise Managed File Transfer, and the Opswise Managed File Transfer components that are required as part of the solution presented by each feature.

- Describes how each feature fits into the Opswise Managed File Transfer business solution.
- Illustrates example solutions of how each feature can be implemented.
- Identifies the Opswise Managed File Transfer components used as part of each solution.
- Provides links to Reference Guides for the components.

Information pertaining to both Opswise Managed File Transfer and Opswise Universal Agent is located in the Opswise Universal Agent 5.2.0 User Guide.

Opswise Managed File Transfer 5.2.0 User Guide

The Opswise Managed File Transfer 5.2.0 User Guide contains *user* information specific to Opswise Managed File Transfer:

- Opswise Managed File Transfer - Overview
- Opswise Managed File Transfer - Components
- Opswise Managed File Transfer - Transferring Files to and from Remote Systems
- Opswise Managed File Transfer - Remote Execution
- Opswise Managed File Transfer - Remote Execution for SAP Systems
- Opswise Managed File Transfer - Web Services Execution

Related Documentation

Opswise Managed File Transfer, as a service of Opswise Universal Agent, is shared many Opswise Universal Agent features and functionality.

Opswise Universal Agent 5.2.0 User Guide

The Opswise Universal Agent 5.2.0 User Guide contains information common to Opswise Managed File Transfer and Opswise Universal Agent:

- Opswise Universal Agent - Copying Files to and from Remote Systems
- Opswise Universal Agent - Encryption
- Opswise Universal Agent - Configuration Management
- Opswise Universal Agent - Component Management
- Opswise Universal Agent - Event Monitoring and File Triggering
- Opswise Universal Agent - Monitoring and Alerting
- Opswise Universal Agent - Messaging and Auditing
- Opswise Universal Agent - Message Translation
- Opswise Universal Agent - Windows Event Log Dump
- Opswise Universal Agent - zOS Cancel Command Support

Opswise Universal Agent 5.2.0 Reference Guide

The Opswise Universal Agent 5.2.0 Reference Guide contains technical information that is both:

- Common to Opswise Universal Agent and Opswise Managed File Transfer
- Specific to either Opswise Universal Agent or Opswise Managed File Transfer

Opswise Managed File Transfer - Overview

Opswise Managed File Transfer

Opswise Managed File Transfer is the Stonebranch Inc. business solution for managed file transfer.

In addition to the basic features inherent in the managed file transfer of files between servers and applications - security, visibility, manageability, reliability, and compliance - Opswise Managed File Transfer provides additional features for managed file transfer.

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

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

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

Usage

The managed file transfer of data provided by Opswise Managed File Transfer lets you streamline business processes by optimizing the integration of file transfers with your business processes. This helps you avoid delays and maximize revenue.

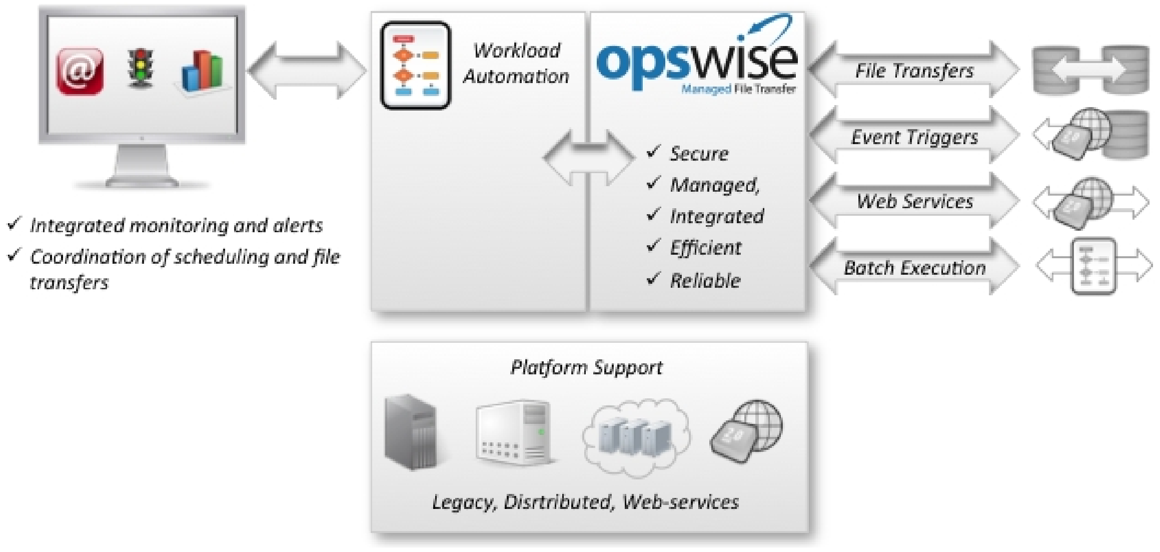
Using Opswise Managed File Transfer enables you to securely transfer files to external partners without disruption of their current business processes. The integration capabilities and ease of use provided by Opswise Managed File Transfer enable you to manage thousands of servers with minimum interaction.

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

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

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

Opswise Managed File Transfer



Implementation

Opswise Managed File Transfer provides simplified implementation, enabling rapid deployment throughout any environment. A common infrastructure and scripting language means that deployments are not platform-specific.

User access to servers and files is managed via operating system security. Also, user access to Opswise Managed File Transfer is centrally managed. All Opswise Managed File Transfer installation materials and documentation are delivered electronically via the customer area of the Stonebranch, Inc. website. This ensures that customers can always access the most current versions and documentation.

All Opswise Managed File Transfer functions and components are delivered in a single installation package for each platform. Native operating system packaging simplifies installation. Opswise Managed File Transfer license keys are not CPU-specific. This simplifies deployment and ensures business continuity.

Stonebranch offers several programs to assist in implementation. These programs are targeted to help organizations implement the solution quickly in order to obtain the fastest return on investment. They include education, implementation, migration and consulting services. See our website at http://www.stonebranch.com/services.html for more information.

Opswise Managed File Transfer - Features

Opswise Managed File Transfer Features

The features that make Opswise Managed File Transfer an intelligent file transfer solution encompass a variety of core and supporting functionality.

The following text describes these features and provides links to detailed information about each one in this document. This includes examples that illustrate feature implementation and links to detailed technical information about the Opswise Managed File Transfer - Components used in that implementation.

The core feature of Opswise Managed File Transfer is Transferring Files to and from Remote Systems in a manner that is both secure and efficient. Transfer sessions can be initiated between the machine initiating the transfer and a remote machine, or between two remote machines.

Elaborate Event Monitoring and File Triggering functionality enables the monitoring local and remote system events, and permits execution of system commands or scripts based on the outcome of the events.

Web Services Execution enables Opswise Managed File Transfer to create file-based events from inbound Internet and message-based application messages, and then write the events to file, thus integrating those applications with Opswise Managed File Transfer system management.

For Opswise Managed File Transfer 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.

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

The Monitoring and Alerting feature of Opswise Managed File Transfer provides for monitoring the status and activity of all Opswise Managed File Transfer 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 Opswise Managed File Transfer's Universal Configuration Manager graphical user interface.

Additionally, Opswise Managed File Transfer offers various methods for the Configuration Refresh of all component data. Opswise Managed File Transfer 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.

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

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

Opswise Managed File Transfer's Remote Execution permits the execution of system commands on remote machines. Additionally, Remote Execution for SAP Systems permits SAP events to be executed on remote SAP systems.

Opswise Managed File Transfer - Components

Overview

Opswise Managed File Transfer 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.

This page provides a description of each component that comprises Opswise Managed File Transfer. Each description provides links to the technical documentation (Reference and Quick Reference Guides) specific to that component. Reference Guides provide detailed technical information about the usage, syntax, format, and values of component commands and configuration options, as well as other information specific to the component. Quick Reference Guides provide summary information on the usage, syntax, format, and values of component commands or configuration options.

Stonebranch also provides separate documents for the installation of operating system-specific component packages and for component-specific error messaging. For links to these documents, see Additional Documentation.

Universal Data Mover

Universal Data Mover is the core component for Opswise Managed File Transfer's managed file transfer functionality. In a secure and automated manner, it allows you to transfer data between any platforms in your environment and initiated from any platform.

Every Universal Data Mover transfer operation is comprised of three components: manager, primary server, and secondary server. The manager receives commands from the user through an interactive session and/or an external script file. It then establishes a transfer session, invoking the primary and secondary servers, which actually conduct the transfer operations. Data is transferred between the servers, with either able to act as the source in a transfer operation.

A transfer session can either be two-party or three-party:

- In a two-party transfer session, the manager also serves as the primary transfer server. Transfer operations occur between the manager/primary server and the secondary server.
- In a three-party transfer session, the manager acts solely as a control point for transfer operations, sending commands to the primary and secondary servers to be executed. Transfer operations take place between the two machines under which these servers are running.

The extensive integration capabilities of Universal Data Mover allow data to be pre- and post-processed.

Universal Data Mover exceeds current security and auditing requirements, including SOX, GLBA, and HIPAA. It supports the most modern security standards and methodology, including SSL encryption, X.509 certificates, and proxy certificates.

If there is a connection failure, Universal Data Mover ensures that all interrupted transfers resume without manual intervention. It integrates with your existing workload management solution to issue alerts if connections are not reestablished after an acceptable time interval.

Technical Documentation

For detailed information on Universal Data Mover, see the following documents:

Universal Data Mover 5.2.0 Reference Guide

Universal Data Mover 5.2.0 Quick Reference Guide

Universal Event Monitor

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

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

Universal Event Monitor detects file creation in real-time on the operating system level and invokes a "handler" to take action on every file matching predefined criteria – whether it is renaming it, processing it locally, moving the file to another server, or notifying your job scheduling system to initiate further processing. It provides rule-based alerts and notifications that enable you to immediately handle any issues that may arise.

Universal Event Monitor can run in either of two-modes: demand-driven or event-driven.

In demand-driven mode, the Universal Event Monitor manager provides the Universal Event Monitor server with event definitions and event handlers, which is a command or script that the server executes based on the outcome of the event. This can be initiated from any system running Opswise Managed File Transfer and scheduled through your scheduling engine.

In event-driven mode, a server monitors one or more system events simultaneously based on event definitions stored in its event definition database. The server monitors each event until it is no longer active, or until the event-driven server ends.

Universal Event Monitor supports the most modern security standards and methodology, including SSL encryption.

UEMLoad

The UEMLoad utility handles all event definition and event handler database management tasks, including adds, updates, deletes, and lists / exports. UEMLoad forwards database requests to a UEM Server, which validates the information.

Technical Documentation

For detailed information on Universal Event Monitor and UEMLoad, see the following documents:

Universal Event Monitor 5.2.0 Reference Guide

Universal Event Monitor 5.2.0 Quick Reference Guide

UEMLoad 5.2.0 Quick Reference Guide

Universal Event Monitor for SOA

Universal Event Monitor for SOA – the SOA "Listener" – lets 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.

Note
Universal Command Agent for SOA – the SOA "Publisher" – is a workload execution component available for use with Opswise Managed File Transfer.

Technical Documentation

For detailed information on Universal Event Monitor for SOA, see the following document:

Universal Event Monitor for SOA 5.2.0 Reference Guide

Universal Enterprise Controller

Universal Enterprise Controller (UEC) provides alerts for activity and availability of the Opswise Universal Agent components 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. These alerts can be picked up by your automation tool and used to pause the submission of jobs and file transfers for nodes that are unavailable, and resume submission once network connectivity or system availability has been reestablished, without manual intervention. You can route these alerts to your existing automation console. This allows for a simple, quick and comprehensive integration, as these systems can remain unchanged when additional agents are added to your infrastructure.

Universal Enterprise Controller installs on a single, central platform, providing the management layer that enables the Universal Event Subsystem, I-Activity Monitor, I-Management Console, and I-Administrator to centralize visibility and management of your workload infrastructure.

UECLoad

UECLoad is a command line application that permits Universal Enterprise Controller users to add, delete, and view Agents in the Universal Enterprise Controller database.

Via UECLoad, a user can add or delete individual Agents, or supply an Agents definition file (**deffile**) with definitions to be added or deleted from Universal Enterprise Controller. UECLoad also can be used to export audit and history records created with the Universal Event Subsystem to multiple formats including text, html, and csv.

Universal Event Subsystem

The Universal Event Subsystem (UES) records, routes, and manages event messages generated by Opswise Universal Agent 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.

Technical Documentation

For detailed information on Universal Enterprise Controller, UECLoad, and the Universal Event Subsystem, see the following documents:

Universal Enterprise Controller 5.2.0 Reference Guide

Universal Event Subsystem 5.2.0 Event Definitions

UECLoad 5.2.0 Quick Reference Guide

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 Opswise Universal Agent infrastructure:

- I-Activity Monitor
- I-Management Console
- I-Administrator

I-Activity Monitor

The I-Activity Monitor client application provides you with end-to-end visibility of workload management activity throughout your Opswise Universal Agent environment.

It provides a graphical user interface for displaying information about the current status and posted alerts for all Agents and SAP systems being monitored by Universal Enterprise Controller.

Whether the workload consists of regular jobs, scripts or commands, I-Activity Monitor lets you see where all processes are executed, as well as when, where, and how they were initiated.

I-Activity Monitor also identifies Opswise Managed File Transfer file transfer jobs, the current state of each transfer, and every instruction executed in a file transfer script. This enables you to know exactly which files have been transferred and which files are still pending.

In addition, I-Activity Monitor can display activity regardless of whether it was initiated by a scheduling system, workflow engine, business application, or end user.

I-Management Console

The I-Management Console client application provides a graphical user interface for remote configuration of all Agents in an enterprise. From a single machine, you can configure a single Agent's components or, simultaneously, multiple Agents.

With I-Management Console, you can define standard security access and authentication policies and ensure that they are active across all servers. You can define which users are allowed to change the policies. An audit log lets you determine when changes were made – and who made them.

I-Management Console lets you distribute configuration information to any server, regardless of its operating system or Opswise Universal Agent release level. It knows which properties apply for each individual Agent based upon release level and operating system, and will only send the appropriate properties to each Agent.

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.

I-Administrator also lets you maintain Universal Enterprise Controller users and their permissions.

Technical Documentation

For detailed information on Universal Enterprise Controller Client Applications, see the following document:

Universal Enterprise Controller Client Applications 5.2.0 User Guide

Universal Broker

Universal Broker, required on all systems running Opswise Universal Agent, manages Opswise Universal Agent 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.

Technical Documentation

For detailed information on Universal Broker, see the following documents:

Universal Broker 5.2.0 Reference Guide

Universal Automation Center Agent

Universal Automation Center Agent (UAG) provides agent services for Opswise Controller, the Stonebranch, Inc. workload automation solution that performs job scheduling, file transfer, and event monitoring across all server platforms in the enterprise.

UAG enables the Controller to schedule workload, transfer files, and monitor events on an Opswise Universal Agent system, integrating with the Controller to provide distributed, workload automation throughout the enterprise.

UAG automatically starts when the Universal Broker starts and stops when the Universal Broker stops.

Technical Documentation

For detailed information on Universal Automation Center Agent, see the following documents:

Universal Automation Center Agent 5.2.0 Reference Guide

Opswise Message Service

Opswise Message Service (OMS) is the network communication provider between Opswise Controller 5.2.0 and Opswise Universal Agent 5.2.0.

Note
Between the 5.2.0 Controller and 5.1.x Agents, and the 5.1.x Controller with 5.2.0 Agents, the 5.1.0 Outboard (Message Hub and Transporter) must be used as their network communications provider.

OMS can be configured to automatically start/restart when the Universal Broker starts/restarts and stop when the Universal Broker stops. It also can be configured to start/restart manually.

Technical Documentation

For detailed information on Opswise Message Service, see the following document:

Opswise Message Service 5.2.0 Reference Guide

Opswise Command Line Interface

Opswise Command Line Interface (CLI) is a set of commands that perform specific actions in an Opswise Controller for executing work on an Agent.

Technical Documentation

For detailed information on Opswise Command Line Interface, see the following documents:

Opswise Controller Remote Interfaces

Opswise Universal Agent Utilities

Opswise Universal Agent Utilities perform a variety of functions for one or more operating systems (some utilities are operating-system specific).

Universal Certificate

Opswise Universal Agent 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 Opswise Universal Agent, if your organization has not yet established one, you can use Universal Certificate to create digital certificates and private keys.

Universal Control

Universal Control provides the ability to start and stop Opswise Universal Agent 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, tailored specifically for Stonebranch databases, allows you to dump one or more databases for back-up and restore purposes.

Universal Database Load

Universal Database Load, tailored specifically for Stonebranch databases, allows you to restore a database that has been previously dumped.

Universal Display Log File

Universal Display Log File is available for the IBM i operating system only, lets you read job log files, write them to standard out, and, optionally, delete the files after read.

Universal Encrypt

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

Although all Opswise Universal Agent command line options can be encrypted using Universal Encrypt, most organizations use it to encrypt and store authentication credentials such as **userid** or **password**. The encrypted command file can be decrypted only by Stonebranch product programs. No decrypt command is provided to decrypt the command file.

Universal Event Log Dump

Universal Event Log Dump (UELD) lets you select records from one of the Windows event logs and write 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 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 Message Translator

Universal Message Translator (UMET) translates error messages 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.

Universal Install Merge

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

UPIMERGE runs automatically during an Opswise Universal Agent installation upgrades on UNIX and Windows. During the install, UPIMERGE combines options and values from existing configuration and component definition files with the options and values in the most recent versions of those files (delivered with the distribution package). The result of each merge is a single file, with preserved options and values residing alongside any new options and values that were introduced to support new Opswise Universal Agent features.

Universal Query

Universal Query (UQUERRY) queries any Universal Broker for Broker-related and active component-related information. You can issue Universal Query from any Opswise Universal Agent installation to query any Universal Broker in the Stonebranch infrastructure.

Universal Return Code

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

The return code of a Windows batch script is the return code of the last command executed. You can use Universal Return Code as the last command to set the return code of the batch script to something different than the return code of the last command executed.

Universal Spool List

Universal Spool List lets you 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 lets you remove component records from the Stonebranch databases. However, you should use Universal Spool Remove only at the direction of Stonebranch, Inc. Customer Support.

Universal Submit Job

Universal Submit Job (USBMJOB) is a command for the IBM i operating system that encapsulates the IBM Submit Job (SBMJOB) command.

Universal Submit Job builds on the functionality of SBMJOB by providing a job submission command that better suits the needs of a remote user issuing IBM i commands via Opswise Universal Agent.

Universal Write-to-Operator

Universal Write-to-Operator (UWTO) is a command line utility for the z/OS UNIX System Services (USS) environment.

Universal Write-to-Operator lets you issue two types of messages to z/OS consoles:

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

Technical Documentation

For detailed information on Opswise Universal Agent Utilities, see the following documents:

Opswise Universal Agent Utilities 5.2.0 Reference Guide

Universal Certificate 5.2.0 Quick Reference Guide

Universal Control 5.2.0 Quick Reference Guide

Universal Query 5.2.0 Quick Reference Guide

Opswise Managed File Transfer Limited Use Components

The following Opswise Universal Agent components are included in Opswise Managed File Transfer with limited use through the exec and execsap commands only.

Note
For detailed information on these limited use components, see the Opswise Universal Agent 5.2.0 User Guide.

Universal Command

Universal Command, the core component for Opswise Universal Agent'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. Any type of program, command, or script file that can be run from the command line interface can be run by Universal Command.

If Universal Command is on the same system as Universal Data Mover, you can execute system commands on remote machines using the Universal Data Mover exec command. Universal Command is included in Opswise Managed File Transfer only as licensed for this purpose.

Technical Documentation

For detailed information on Universal Command, see the following documents:

Universal Command 5.2.0 Reference Guide

Universal Command 5.2.0 Quick Reference Guide

Universal Command Agent for SOA

Universal Command Agent for SOA – the SOA "Publisher" – lets you extend the workload execution and management features of Opswise Universal Agent 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.

If Universal Command Agent for SOA and Universal Command are on the same system as Universal Data Mover, you can execute system commands on remote machines using the Universal Data Mover exec command. The system commands can, in turn, execute Universal Command Agent for SOA workloads. Universal Command Agent for SOA is included in Opswise Managed File Transfer only as licensed for this purpose.

Technical Documentation

For detailed information on Universal Command Agent for SOA, see the following documents:

Universal Command Agent for SOA 5.2.0 Reference Guide

Getting Started with Universal Command Agent for SOA - MQ Connector 5.2.0

Getting Started with Universal Command Agent for SOA - XD Connector 5.2.0

Universal Connector

Universal Connector *for Use with SAP® ERP* is a command line interface that controls background processing within an SAP system, allowing any computer on a network to manage SAP background processing tasks from any scheduling system on any platform. When Universal Connector is told which SAP system to connect to and what background processing tasks to perform, it connects to that SAP system and processes the request.

If Universal Connector is on the same system as Universal Data Mover, you can execute SAP events using the Universal Data Mover execsap command. Universal Connector is included in Opswise Managed File Transfer only as licensed for this purpose.

Technical Documentation

For detailed information on Universal Connector, see the following documents:

Universal Connector 5.2.0 Reference Guide

Universal Connector 5.2.0 Quick Reference Guide

Additional Documentation

In addition to component-specific documentation, Stonebranch also provides the following documentation for Opswise Universal Agent:

Installation and Administration

Opswise Universal Agent 5.2.0 Installation and Administration Documentation

Opswise Universal Agent 5.2.0 Installation Requirements and Summary

Opswise Universal Agent 5.2.0 Installation Quick Start Guides

Messages and Codes

Opswise Universal Agent 5.2.0 Messages and Codes

Opswise Managed File Transfer - Transferring Files to and from Remote Systems

Transferring Files to and from Remote Systems

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

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

Transfer Operation Components

There are three components to any Opswise Managed File Transfer transfer operation:

1. Manager
2. Primary server
3. Secondary server

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

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

Manager

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

Primary Server

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

Secondary Server

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

Transfer Sessions

Overview

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

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

Logical Names

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

Two-Party Transfer Sessions

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

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

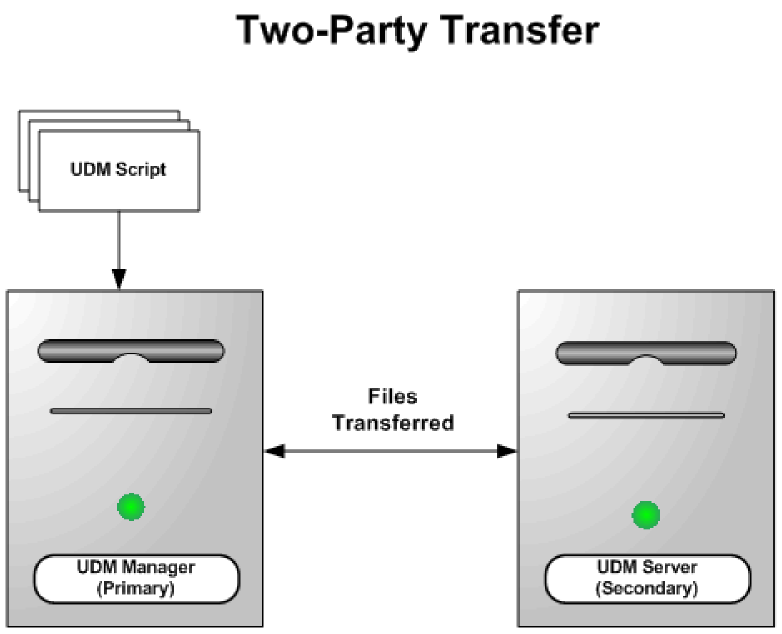
(See the following illustration.)

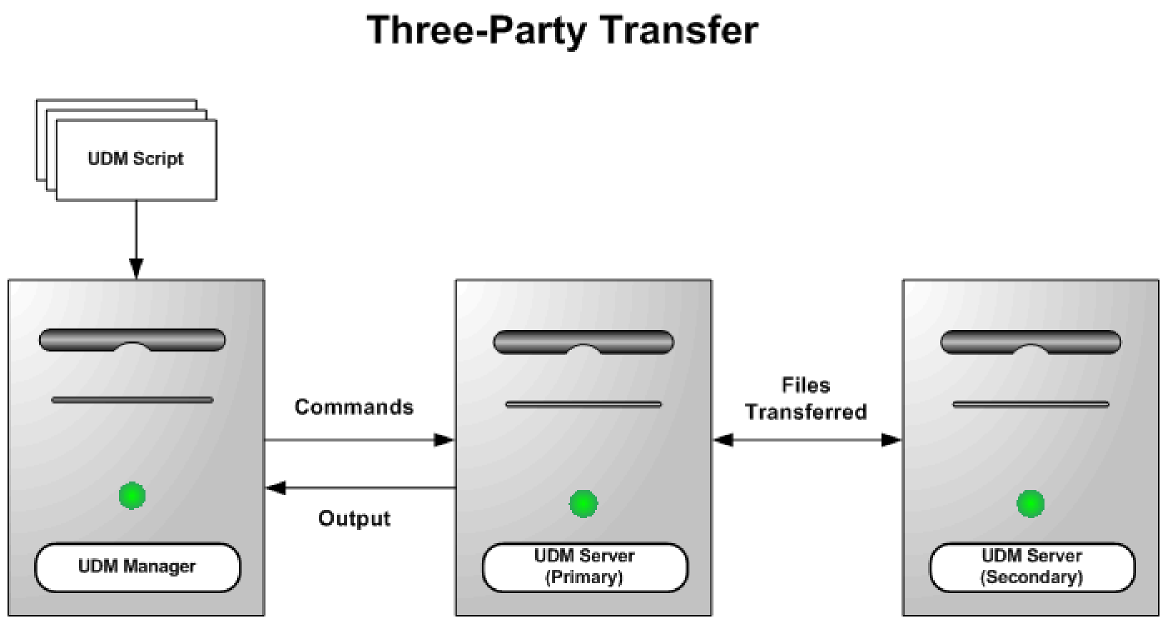
Three-Party Transfer Sessions

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

(See the following illustration.)

Transfer Sessions (Illustrated)


Two-Party Transfer
UDM Script
Files Transferred
UDM Manager (Primary)
UDM Server (Secondary)


Three-Party Transfer
UDM Script
Commands
Output
Files Transferred
UDM Manager
UDM Server (Primary)
UDM Server (Secondary)

Transferring Files to and from Remote Systems - Examples

Transferring Files Examples - z/OS

- Copy a File to an Existing z/OS Sequential Data Set
- Copy a File to a New z/OS Sequential Data Set
- Copy a z/OS Sequential Data Set to a File
- Copy a Set of Files to an Existing z/OS Partitioned Data Set
- Copy a Set of Files to a New z/OS Partitioned Data Set

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

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

Transferring Files Examples - Windows and UNIX

- Simple File Copy to the Manager - Windows and UNIX
- Simple File Copy to the Server - Windows and UNIX
- Copy a Set of Files - Windows and UNIX

These examples illustrate two-party transfer sessions.

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

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

Transferring Files Examples - IBM i

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

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

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

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

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

Copy a File to an Existing zOS Sequential Data Set

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

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

DD File System

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

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

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

DSN File System

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

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

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

Components

Universal Data Mover Manager for z/OS

Copy a File to a New zOS Sequential Data Set

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

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

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

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

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

DSN File System

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

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

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

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

Components

Universal Data Mover Manager for z/OS

Copy a zOS Sequential Data Set to a File

Copy a z/OS Sequential Data Set to a File

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

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

DD File System

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

DSN File System

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

Components

Universal Data Mover Manager for z/OS

Copy a Set of Files to an Existing zOS Partitioned Data Set

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

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

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

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

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

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

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

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

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

DD File System

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

DSN File System

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

Components

Universal Data Mover Manager for z/OS

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

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

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

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

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

DSN File System

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

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

Components

Universal Data Mover Manager for z/OS

Simple File Copy to the Manager - Windows and UNIX

Simple File Copy to the Manager

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

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

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

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

Components

Universal Data Mover Manager for Windows

Universal Data Mover Manager for UNIX

Simple File Copy to the Server - Windows and UNIX

Simple File Copy to the Server

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

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

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

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

Components

Universal Data Mover Manager for Windows

Universal Data Mover Manager for UNIX

Copy a Set of Files - Windows and UNIX

Copy a Set of Files

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

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

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

The following files will be created on the destination machine:

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

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

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

Components

Universal Data Mover Manager for Windows

Universal Data Mover Manager for UNIX

Copy a File to an Existing IBM i File

Copy a File to an Existing IBM i File

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

LIB File System

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

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

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

HFS File System

```
1 set _echo=yes
2 set _halton=warn
3 open unix=sol9 user=top098 pwd=p100m
4 filesys local=hfs
5 cd unix=/opt/app/data
6 mode type=text
7 attrib local createop=replace
8 copy unix=data10.txt local=/opt/appdata
9 quit
```

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

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

Components

Universal Data Mover Manager for IBM i

Copy an IBM i Data Physical File to a File

Copy an IBM i Data Physical File to a File

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

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

LIB File System

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

Components

Universal Data Mover Manager for IBM i

Copy a Set of Files to an Existing Data Physical File

Copy a Set of Files to an Existing Data Physical File

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

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

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

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

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

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

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

LIB File System

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

Components

Universal Data Mover Manager for IBM i

Copy a File to a New IBM i Data Physical File

Copy a File to a New IBM i Data Physical File

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

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

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

LIB File System

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

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

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

Components

Universal Data Mover Manager for IBM i

Copy a File to a New IBM i Source Physical File

Copy a File to a New IBM i Source Physical File

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

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

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

LIB File System

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

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

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

Components

Universal Data Mover Manager for IBM i

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

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

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

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

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

LIB File System

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

Components

Universal Data Mover Manager for IBM i

Copy Different Types of IBM i Files Using forfiles and $(_file.type)

Copy Different Types of IBM i Files using forfiles and $(_file.type)

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

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

LIB File System

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

Components

Universal Data Mover Manager for IBM i

Invoke a Script from an IBM i Batch Job

Invoke a Script from an IBM i Batch Job

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

The following example illustrates both:

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

LIB file system

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

Components

Universal Data Mover Manager for IBM i

Opswise Managed File Transfer - Remote Execution

Remote Execution via UDM

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

These pages provide information on the remote execution features and functionality of the Opswise Managed File Transfer business solution.

Opswise Managed File Transfer provides access to Universal Command remote execution via the Universal Data Mover exec command. The exec command invokes the Universal Command Manager and provides parameters for passing a subset of the Universal Command Manager options.

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

Remote Execution Components

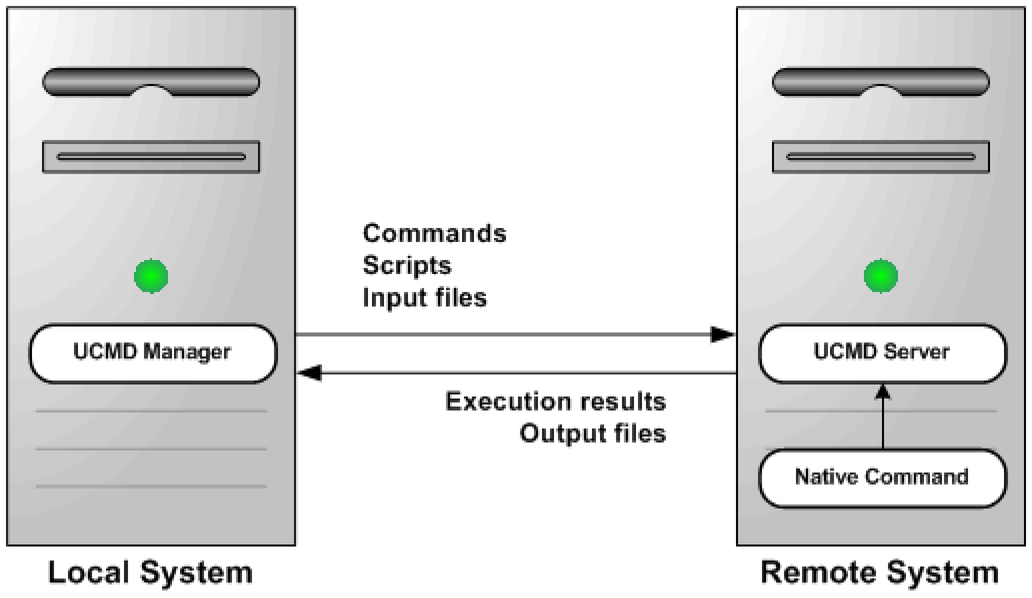
Opswise Managed File Transfer Remote Execution using Universal Command consists primarily of two Opswise Universal Agent components:

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

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

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

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



Remote Execution via UDM - Primer

Overview

This page discusses the basics of how to execute remote work using Opswise Managed File Transfer via Universal Data Mover (UDM).

Note
Please read Remote Execution via UDM - Overview prior to reading this page, which builds upon the material presented in the Overview.

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

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

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

Remote Execution Examples - exec Command Parameters

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

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

Remote Execution Requirements

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

The platform-independent nature of the UDM scripting language means that the format of exec is the same regardless of the UDM Manager's host platform. See the Universal Data Mover 5.2.0 Reference Guide for platform-specific information on executing UDM Manager, using UDM script files, and invoking exec.

exec instructs UDM to spawn a Universal Command (UCMD) Manager process. The Universal Command 5.2.0 Reference Guide contains platform-specific information for invoking UCMD Manager. A UCMD Server installed on the remote system receives the command specified by exec's **cmd** parameter and executes it.

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

For information on securing access to Opswise Universal Agent components, see Security.

Executing the Examples

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

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

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

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

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

Remote Execution Requirements for z/OS

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

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

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

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

Remote z/OS Execution Using an IP Address

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

Remote z/OS Execution Using a Host Name

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

Remote z/OS Execution Using a UDM Logical Session Name

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

Remote Execution Requirements for Windows

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

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

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

Remote Windows Execution Using an IP Address

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

Remote Windows Execution Using a Host Name

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

Remote Windows Execution Using a UDM Logical Session Name

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

Remote Execution Requirements for UNIX

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

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

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

Remote UNIX Execution Using an IP Address

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

Remote UNIX Execution Using a Host Name

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

Remote UNIX Execution Using a UDM Logical Session Name

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

Remote Execution Requirements for IBM i

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

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

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

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

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

Remote IBM i Execution Using an IP Address

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

Remote IBM i Execution Using a Host Name

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

Remote IBM i Execution Using a UDM Logical Session Name

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

Remote Execution via UDM - Examples

Remote Execution Examples

- Windows Directory Listing Using a Batch File - Default Directory
- Windows Directory Listing Using a Batch File - Returned File
- UNIX - Listing Using a Shell Script
- UNIX - Integrating UDM with FTP Using a Shell Script
- UNIX - Integrating UDM with FTP Using a Command Reference
- IBM i from Windows, UNIX, or IBM i - exec Command Return Codes

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

Windows Directory Listing Using a Batch File - Default Directory

Windows Directory Listing Using a Batch File - Default Directory

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

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

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

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

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

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

Output sent to **stdout.txt**.

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

 Directory of C:\wrk\xmp\win

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

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

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

UDM exec Command Parameters

The exec command parameters used in this example are:

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

Components

Universal Data Mover Manager for Windows

Universal Command Server for Windows

Windows Directory Listing Using a Batch File - Returned File

Windows Directory Listing Using a Batch File - Returned File

This example builds on the example illustrated in Windows Directory Listing Using a Batch File - Default Directory.

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

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

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

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

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

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

Output sent to **stdout.txt**.

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

 Directory of C:\wrk\xmp\win

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

UDM exec Command Parameters

The exec command parameters used in this example are:

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

Components

Universal Data Mover Manager for Windows

Universal Command Server for Windows

UNIX - Listing Using a Shell Script

UNIX Listing Using a Shell Script

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

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

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

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

UDM Script Explanation

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

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

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

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

Output sent to **stdout.txt**.

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

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

```
2011.07.28 10.13.06.845 UNV2800I Universal Data Mover 5.2.0 Level 0 Release Build 104 started.
2011.07.28 10.13.06.845 Processing script: ls.udm
2011.07.28 10.13.06.847 open r=houston user=joe pwd=* port=7887
2011.07.28 10.13.07.114 Data session established using cipher: NULL-MD5
2011.07.28 10.13.07.159 Two party session established with r (component 1278600806)
2011.07.28 10.13.07.161 Transfer mode settings:
2011.07.28 10.13.07.198   type=binary
2011.07.28 10.13.07.198   trim=no
2011.07.28 10.13.07.198 Session options:
2011.07.28 10.13.07.198   Keep Alive Interval:    120
2011.07.28 10.13.07.198   Network Fault Tolerant: yes
2011.07.28 10.13.07.198 exec r cmd="sh /home/joe/wrk/xmp/ls/ls.sh" user=joe pwd=* port=7887
ls.sh
stdout.txt

2011.07.28 10.13.08.072 quit
2011.07.28 10.13.08.074 Session closed
2011.07.28 10.13.08.074 Finished processing script: ls.udm
2011.07.28 10.13.10.074 UNV2801I Universal Data Mover 5.2.0 Level 0 Release Build 104 ended
successfully.
```

UDM exec Command Parameters

The exec command parameters used in this example are:

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

Components

Universal Data Mover Manager for UNIX

Universal Command Server for UNIX

UNIX - Integrating UDM with FTP Using a Shell Script

UNIX - Integrating UDM with FTP Using a Shell Script

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

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

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

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

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

The shell script sets up and executes FTP commands.

```
ftp -ipnv houston <
```

UDM exec Command Parameters

The exec command parameters used in this example are:

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

Components

Universal Data Mover Manager for UNIX

Universal Command Server for UNIX

UNIX - Integrating UDM with FTP Using a Command Reference

UNIX - Integrating UDM with FTP Using a Command Reference

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

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

If you are not familiar with Command References, please read Command References.

UDM Script Explanation

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

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

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

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

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

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


ftp -ipnv $1 <
```

UDM exec Command Parameters

The exec command parameters used in this example are:

Parameter	Description
cmdref	Command Reference file name and, optionally, options to be passed to the command or script.

user	Remote user ID with which to authenticate and execute the command on the remote system.
pwd	Password with which to authenticate the user ID on the remote system.
port	Port that the Universal Broker is listening on for the remote machine. The port is inherited from the UDM Manager's configuration file unless explicitly overridden in the call to the exec command.

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

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

This example demonstrates using the _execrc built-in variable.

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

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

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

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

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

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

UDM Script Explanation

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

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

Operating System-Specific Information

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

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

UDM exec Command Parameters

The exec command parameters used in this example are:

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

Components

Universal Data Mover Manager for IBM i

Universal Data Mover Manager for Windows

Universal Data Mover Manager for UNIX

Universal Command Manager for IBM i

Opswise Managed File Transfer - Remote Execution for SAP Systems

Remote Execution for SAP Systems via UDM

These pages provides information on the Remote Execution for SAP feature and functionality of the Opswise Managed File Transfer business solution.

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

Opswise Managed File Transfer provides access to Universal Connector remote execution via the Universal Data Mover (UDM) execsap command. The execsap command invokes Universal Connector and executes SAP events on remote machines if you have Universal Connector on the same system with the Universal Data Mover Manager.

Remote Execution for SAP Examples

The remote execution for SAP examples illustrated in these pages are specific to the operating systems supported by Opswise Universal Agent for the Remote Execution for SAP feature of Opswise Managed File Transfer. The examples demonstrate the use of Universal Connector *for Use with SAP® ERP* to define SAP jobs.

Links to detailed technical information on appropriate Universal Managed File Transfer components are provided for each example.

Remote Execution for SAP Systems via UDM - Examples

Remote Execution for SAP Systems Examples

These examples illustrate the Remote Execution of SAP feature of Opswise Managed File Transfer using the Universal Data Mover execsap command.

Raising an SAP Event for zOS Example

Raising an SAP Event for z/OS Example

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

In this example, we assume the following scenario:

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

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

Raising an SAP Event for z/OS JCL

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

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

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

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

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

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

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

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

close
/*
```

Components

Universal Data Mover Manager for z/OS

Universal Data Mover Server for UNIX

Universal Connector for z/OS

Raising an SAP Event for UNIX Example

Raising an SAP Event for UNIX Example

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

In this example, we assume the following scenario:

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

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

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

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

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

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

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

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

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

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

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

close
```

Components

Universal Data Mover Manager for UNIX

Universal Data Mover Server for zOS

Universal Connector for zOS

Opswise Managed File Transfer - Web Services Execution

Web Services Execution (Inbound Implementation)

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

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

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

Universal Event Monitor (UEM) monitors one or more local or remote system events. It also can execute a system command or script based on the outcome of the events that it is monitoring.

Web Services Execution (Inbound Implementation) - Examples

Web Services Execution Examples

- Inbound Implementation - JMS
- Inbound Implementation - SOAP

Inbound Implementation - JMS

Inbound Implementation - JMS

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

The following figure illustrates an example of this construction.

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

In the following examples:

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

ActiveMQ Topic

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

```
<sb:JMSConnection>
        <sb:Name>JMS ActiveMQ Topic Listener - soatest2/</sb:Name>
        <sb:InitialContextProperties>
            <sb:Property>
                <sb:Name>java.naming.factory.initial</sb:Name>

<sb:Value>org.apache.activemq.jndi.ActiveMQInitialContextFactory</sb:Value>
            </sb:Property>
            <sb:Property>
                <sb:Name>java.naming.provider.url</sb:Name>
                <sb:Value>tcp://soatest2:61616</sb:Value>
            </sb:Property>
        </sb:InitialContextProperties>
        <sb:ConnectionFactory>ConnectionFactory</sb:ConnectionFactory>
        <sb:Listeners>
            <sb:JMSListener>

<sb:Destination>dynamicTopics/UemsoaStartTopicA</sb:Destination>
                <sb:Actions>
                    <sb:JMSFileWriter>
                        <sb:Directory>filesystem</sb:Directory>
<sb:FilenamePattern>ActiveMQ_Topic_%Seq%.txt</sb:FilenamePattern>
                        <sb:StartSequenceNumber>1</sb:StartSequenceNumber>
                        <sb:WriteProperties>false</sb:WriteProperties>
                    </sb:JMSFileWriter>
                </sb:Actions>
            </sb:JMSListener>
        </sb:Listeners>
    </sb:JMSConnection>
</pre>
```

Websphere Queue

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

```
<sb:JMSConnection>
       <sb:Name>JMS WebSphere Queue Listener - soatest2</sb:Name>
       <sb:InitialContextProperties>
           <sb:Property>
               <sb:Name>java.naming.factory.initial</sb:Name>

<sb:Value>com.ibm.websphere.naming.WsnInitialContextFactory</sb:Value>
            </sb:Property>
            <sb:Property>
                <sb:Name>java.naming.provider.url</sb:Name>
                <sb:Value>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.

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

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

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

Event Options

The Event options used in this example are:

Option	Description
event_id	Identifier that uniquely identifies an event definition record.
event_type	Type of system event represented by the event definition record.
comp_name	Event-driven UEM Server responsible for monitoring the event.
state	Event definitions that should be processed or ignored by UEM.
tracking_int	Event definitions that should be processed or ignored by UEM.
triggered_id	ID of an event handler record that UEM will execute when an event occurrence is triggered.
filespec	Name of a file to monitor.
min_file_size	Size a file must be in order to be considered complete by UEM.
rename_file	Specifies whether or not UEM should rename a monitored file when an event occurrence is triggered.
rename_filespec	Specifies how a file should be renamed when an event occurrence is triggered.
handler_id	Identifier that uniquely identifies an event handler record.

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.

Contents of File udm.script

```
open dest_server=192.168.1.1 user=qatest pwd=qatest
attrib dest_server createop=replace
forfiles local=2*.xml
   copy local=$(_file)
end
close
```

Components

Universal Event Monitor

UEMLoad

Universal Event Monitor for SOA

Inbound Implementation - SOAP

Inbound Implementation - SOAP

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