

Universal Agent 7.4.x

Universal Agent 7.4.x User Guide

© 2023 by Stonebranch, Inc. All Rights Reserved.

1. Universal Agent 7.4.x User Guide . . . 7
1.1 Universal Command Overview . . . 7
1.2 Universal Data Mover Overview . . . 10
1.3 Universal Agent Features . . . 11
1.4 Universal Agent Components . . . 12
1.5 Remote Execution via Universal Command and Universal Data Mover . . . 20
1.5.1 Universal Command - Remote Execution . . . 20
1.5.1.1 Remote Execution via Universal Command - Primer . . . 21
1.5.1.2 Remote Execution via Universal Command - Examples . . . 24
1.5.1.2.1 Back up UNIX Directory to zOS Dataset . . . 26
1.5.1.2.2 Restore UNIX Directory Backup from zOS Dataset to UNIX Directory . . . 27
1.5.1.2.3 Directory Listing for UNIX Server from zOS . . . 28
1.5.1.2.4 Directory Listing for Windows Server from zOS . . . 29
1.5.1.2.5 Provide Network Status of Remote UNIX from zOS . . . 29
1.5.1.2.6 Use UNIX tee Command to Store stdout to Local Server and zOS . . . 30
1.5.1.2.7 Use an Encrypted Command File for User ID and Password on zOS . . . 31
1.5.1.2.8 Override Standard zOS IO File ddnames . . . 32
1.5.1.2.9 Override zOS Standard Files with Procedure Symbolic Parameters . . . 33
1.5.1.2.10 Specifying UCMD for zOS Options with the EXEC PARM . . . 34
1.5.1.2.11 Executing an Existing Windows .bat File from zOS . . . 35
1.5.1.2.12 Using Manager Fault Tolerance from zOS . . . 35
1.5.1.2.13 Restarting a Manager Fault Tolerant UCMD Manager on zOS . . . 36
1.5.1.2.14 Automatically Create a Unique zOS Command ID Using CA-Driver Variables . . . 37
1.5.1.2.15 Automatically Create a Unique zOS Command ID Using Zeke Variables . . . 39
1.5.1.2.16 Automatically Create a Unique zOS Command ID Using OPC Variables . . . 40
1.5.1.2.17 Universal Submit Job from zOS to IBM i Using Remote Reply Facility . . . 41
1.5.1.2.18 Executing Universal Return Code within a Script via UCMD Manager for zOS . . . 43
1.5.1.2.19 Executing URC and UMET within a Script via UCMD Manager for zOS . . . 44
1.5.1.2.20 Back up UNIX Directory to Windows . . . 45
1.5.1.2.21 Restore UNIX Directory Backup from Windows to UNIX . . . 46
1.5.1.2.22 Provide Network Status of Remote UNIX from Windows . . . 47
1.5.1.2.23 Redirect Standard Out and Standard Error to Windows . . . 48
1.5.1.2.24 Start UNIX Background Process from Windows . . . 48
1.5.1.2.25 Redirect Standard Input from Initiating System on Windows . . . 49
1.5.1.2.26 Universal Submit Job from Windows to IBM i . . . 50
1.5.1.2.27 Provide Network Status of Remote Windows from UNIX . . . 50
1.5.1.2.28 Redirect Standard Out and Standard Error to UNIX . . . 51
1.5.1.2.29 Redirect Standard Input from Initiating System to UNIX . . . 52
1.5.1.2.30 Redirect Standard Input in UNIX Background Process . . . 52
1.5.1.2.31 Issue Universal Submit Job from UNIX to IBM i . . . 53
1.5.1.2.32 Provide Network Status of Remote Windows from IBM i . . . 53
1.5.1.2.33 Execute Script to Provide Network Status of Remote Windows from IBM i . . . 54
1.5.1.2.34 Display Library with Manager Fault Tolerance Active Using USBMJOB . . . 55
1.5.1.2.35 Universal Submit Job from zOS to IBM i . . . 55
1.5.2 Universal Data Mover - Remote Execution . . . 56
1.5.2.1 Remote Execution via Universal Data Mover - Primer . . . 57
1.5.2.2 Remote Execution via Universal Data Mover - Examples . . . 60
1.5.2.2.1 Windows Directory Listing Using a Batch File - Default Directory . . . 60
1.5.2.2.2 Windows Directory Listing Using a Batch File - Returned File . . . 61
1.5.2.2.3 UNIX - Listing Using a Shell Script . . . 63
1.5.2.2.4 UNIX - Integrating UDM with FTP Using a Shell Script . . . 64
1.5.2.2.5 UNIX - Integrating UDM with FTP Using a Command Reference . . . 65
1.5.2.2.6 IBM i from Windows, UNIX, or IBM i - exec Command Return Codes . . . 66
1.6 Remote Execution for SAP Systems . . . 68
1.6.1 Remote Execution for SAP Systems - Examples . . . 69
1.6.1.1 Define Job, Run Job, Get Output, and Purge Job . . . 69
1.6.1.2 Submitting Job to SAP Using SAP Job as Template - zOS . . . 70
1.6.1.3 Submitting Job to SAP Using Job Definition File - zOS . . . 71
1.6.1.4 Running Job on SAP Using SAP Job - zOS . . . 72
1.6.1.5 Running Job on SAP Using Job Definition File - zOS . . . 73
1.6.1.6 Running an SAP Job on a Specific SAP Server - zOS . . . 75
1.6.1.7 Variant Substitution - zOS . . . 76
1.6.1.8 Creating a Variant Substitution Using GENERATE VARDEF Command - zOS . . . 78
1.6.1.9 Creating a Job Definition Using GENERATE JOBDEF Command - zOS . . . 79
1.6.1.10 Submitting an SAP Job Using SAP Job as Template - UNIX . . . 80
1.6.1.11 Submitting an SAP Job Using Job Definition File - UNIX . . . 81
1.6.1.12 Running an SAP Job Using SAP Job as Template - UNIX . . . 82
1.6.1.13 Running an SAP Job Using a Job Definition File - UNIX . . . 83
1.6.1.14 Running an SAP Job on a Specific SAP Server - UNIX . . . 84
1.6.1.15 Variant Substitution - UNIX . . . 85
1.6.1.16 Creating a Variant Definition Using GENERATE VARDEF Command - UNIX . . . 87
1.6.1.17 Creating Job Definition Using GENERATE JOBDEF Command - UNIX . . . 88

1.6.2 Mass Activities Support Example for zOS . . . 89
1.6.3 Batch Input Monitoring Example for zOS . . . 93
1.6.4 Mass Activities Support in Universal Connector . . . 94
1.6.5 Batch Input Monitoring in Universal Connector . . . 96
1.6.6 Universal Data Mover - Remote Execution for SAP Systems . . . 97
1.6.6.1 Remote Execution for SAP Systems via UDM - Examples . . . 98
1.6.6.1.1 Raising an SAP Event for zOS Example . . . 98
1.6.6.1.2 Raising an SAP Event for UNIX Example . . . 99
1.7 Web Services Execution . . . 100
1.7.1 Universal Agent - Web Services Examples . . . 101
1.7.1.1 Using Universal Agent to Publish to a SOA Workload - Windows and UNIX . . . 102
1.7.1.2 Message Payload for SOAP - Windows and UNIX . . . 105
1.7.1.3 Logging Configuration - Windows and UNIX . . . 106
1.7.1.4 UAC HTTP Form - Windows and UNIX . . . 108
1.7.1.5 Outbound SOAP Implementation - zOS . . . 109
1.7.1.6 Inbound SOAP Implementation - Windows and UNIX . . . 111
1.7.1.7 Inbound JMS Implementation - Windows and UNIX . . . 117
1.7.2 Universal Data Mover - Web Services Execution . . . 120
1.7.2.1 Web Services Execution (Inbound Implementation) - Examples . . . 121
1.7.2.1.1 Inbound Implementation - JMS . . . 121
1.7.2.1.2 Inbound Implementation - SOAP . . . 124
1.8 Copying Files to and from Remote Systems . . . 130
1.8.1 Copy from Local zOS to Remote Windows . . . 131
1.8.2 Copy from Remote Windows to Local zOS . . . 132
1.8.3 Copy from Local zOS to Remote UNIX . . . 133
1.8.4 Copy from Remote UNIX to Local zOS . . . 134
1.8.5 Copy from Local zOS to Remote IBM i . . . 134
1.8.6 Copy from Remote IBM i to Local zOS . . . 135
1.8.7 Third-Party Copy via Local zOS, from Windows to UNIX . . . 136
1.8.8 Third-Party Copy via Local zOS, from UNIX to Windows . . . 138
1.8.9 Third-Party Copy via Local zOS, from Windows to Windows . . . 139
1.8.10 Third-Party Copy via Local zOS, from UNIX to UNIX . . . 141
1.8.11 Copy from Local zOS to Remote System (in Binary) . . . 142
1.8.12 Copy from Remote System to Local zOS (in Binary) . . . 143
1.8.13 Copy from Local zOS to Remote zOS . . . 144
1.8.14 Copy from Remote zOS to Local zOS . . . 145
1.8.15 Copy from Local zOS to Remote Windows (with Windows Date Variables) . . . 146
1.8.16 Copy from Local zOS to Remote UNIX (with UNIX Date Variables) . . . 146
1.8.17 Copy from Remote UNIX to Local zOS Using cat Command . . . 147
1.8.18 Copy from Remote UNIX to Local Windows . . . 148
1.8.19 Copy From Local Windows to Remote UNIX . . . 149
1.8.20 Copy from Remote UNIX to Local Windows Using the UNIX cat Command . . . 150
1.8.21 Copy from Local UNIX to Remote Windows . . . 150
1.8.22 Copy Encrypted File from Local UNIX to Remote Windows . . . 151
1.8.23 Copy from Remote Windows to Local UNIX . . . 151
1.8.24 Copy Encrypted File from Remote Windows to Local UNIX . . . 152
1.8.25 Copy from Remote Windows to Local IBM i via UCMD Manager . . . 153
1.8.26 Copy from Remote IBM i to Local Windows via UCMD Manager . . . 153
1.8.27 Copy from Local Windows to Remote IBM i via UCMD Manager . . . 154
1.8.28 Copy from Local IBM i to Remote Windows via UCMD Manager . . . 155
1.9 Transferring Files to and from Remote Systems . . . 156
1.9.1 Transfer Sessions . . . 156
1.9.2 Transferring Files to and from Remote Systems - Examples . . . 159
1.9.2.1 Copy a File to an Existing zOS Sequential Data Set . . . 159
1.9.2.2 Copy a File to a New zOS Sequential Data Set . . . 160
1.9.2.3 Copy a zOS Sequential Data Set to a File . . . 161
1.9.2.4 Copy a Set of Files to an Existing zOS Partitioned Data Set . . . 162
1.9.2.5 Copy a Set of Files to a New zOS Partitioned Data Set . . . 163
1.9.2.6 Simple File Copy to the Manager - Windows and UNIX . . . 164
1.9.2.7 Simple File Copy to the Server - Windows and UNIX . . . 164
1.9.2.8 Copy a Set of Files - Windows and UNIX . . . 165
1.9.2.9 Copy a File to an Existing IBM i File . . . 165
1.9.2.10 Copy an IBM i Data Physical File to a File . . . 166
1.9.2.11 Copy a Set of Files to an Existing Data Physical File . . . 167
1.9.2.12 Copy a File to a New IBM i Data Physical File . . . 167
1.9.2.13 Copy a File to a New IBM i Source Physical File . . . 168
1.9.2.14 Copy a Set of Files to a New Data Physical File on IBM i . . . 169
1.9.2.15 Copy Different Types of IBM i Files Using forfiles and $(_file.type) . . . 169
1.9.2.16 Invoke a Script from an IBM i Batch Job . . . 170
1.10 Encryption . . . 170
1.10.1 Encryption - Examples . . . 171
1.10.1.1 Creating Encrypted Command File - zOS . . . 172

1.10.1.2 Using Encrypted Command File - zOS . . . 173
1.10.1.3 Creating Encrypted Command File - Windows . . . 174
1.10.1.4 Using Encrypted Command File - Windows . . . 175
1.10.1.5 Creating Encrypted Command File - UNIX . . . 175
1.10.2 Using Encrypted Command File - UNIX . . . 177
1.10.3 Creating Encrypted Command File - IBM i . . . 177
1.10.4 Using Encrypted Command File - IBM i . . . 178
1.11 Configuration Management for Universal Agent . . . 179
1.11.1 Configuration Methods . . . 179
1.11.1.1 Configuration Methods - Command Line . . . 180
1.11.1.2 Configuration Methods - Command File . . . 181
1.11.1.3 Configuration Methods - Environment Variables . . . 182
1.11.1.4 Configuration Methods - Configuration File . . . 182
1.11.2 Remote Configuration . . . 184
1.11.3 Universal Configuration Manager . . . 187
1.11.3.1 Universal Configuration Manager - Universal Application Container Server . . . 190
1.11.3.2 Universal Configuration Manager - Universal Automation Center Agent (UAG) . . . 194
1.11.3.3 Universal Configuration Manager - Universal Broker . . . 201
1.11.3.4 Universal Configuration Manager - Universal Command Manager . . . 224
1.11.3.5 Universal Configuration Manager - Universal Command Server . . . 234
1.11.3.6 Universal Configuration Manager - Universal Connector for SAP . . . 252
1.11.3.7 Universal Configuration Manager - Universal Control Manager . . . 266
1.11.3.8 Universal Configuration Manager - Universal Control Server . . . 273
1.11.3.9 Universal Configuration Manager - Universal Data Mover Manager . . . 288
1.11.3.10 Universal Configuration Manager - Universal Data Mover Server . . . 299
1.11.3.11 Universal Configuration Manager - Universal Event Log Dump . . . 314
1.11.3.12 Universal Configuration Manager - Universal Event Monitor Manager . . . 316
1.11.3.13 Universal Configuration Manager - Universal Event Monitor Server . . . 321
1.11.3.14 Universal Configuration Manager - Universal Message Service (OMS) . . . 332
1.11.3.15 Universal Configuration Manager - Universal Query . . . 340
1.11.4 Configuration Refresh . . . 343
1.11.5 Refreshing via Universal Control Examples . . . 346
1.11.5.1 Refreshing Universal Broker from zOS . . . 346
1.11.5.2 Refreshing a Component from zOS . . . 347
1.11.5.3 Refreshing Universal Broker from Windows . . . 348
1.11.5.4 Refreshing a Component from Windows . . . 349
1.11.5.5 Refreshing Universal Broker from UNIX . . . 349
1.11.5.6 Refreshing a Component from UNIX . . . 350
1.11.5.7 Refreshing Universal Broker from IBM i . . . 351
1.11.5.8 Refreshing a Component from IBM i . . . 351
1.11.6 Merging Configuration Options . . . 352
1.11.6.1 Files Used in UPI Merge Examples . . . 353
1.11.6.2 Merge Configuration Files Using Program Defaults . . . 354
1.11.6.3 Merge Configuration Files Introducing New Options . . . 355
1.11.6.4 Merge Configuration Files Using Installation-Dependent Values . . . 355
1.11.7 Configuration Options . . . 356
1.12 Component Management . . . 357
1.12.1 Component Definition . . . 358
1.12.2 Component Definition Options . . . 358
1.12.3 Starting and Stopping Agent Components . . . 358
1.12.4 Starting and Stopping Agent Components - Examples . . . 359
1.12.4.1 Starting and Stopping Universal Broker - z/OS . . . 360
1.12.4.2 Starting Universal Broker - Windows . . . 361
1.12.4.3 Starting Universal Broker - UNIX . . . 364
1.12.4.4 Starting, Ending, Working with Universal Broker - IBM i . . . 367
1.12.4.5 Starting and Stopping Universal Enterprise Controller - z/OS . . . 368
1.12.4.6 Starting and Stopping Universal Enterprise Controller - Windows . . . 369
1.12.4.7 Starting a z/OS Component via Universal Control . . . 369
1.12.4.8 Stopping a z/OS Component via Universal Control . . . 370
1.12.4.9 Starting a Windows Component via Universal Control . . . 371
1.12.4.10 Stopping a Windows Component via Universal Control . . . 371
1.12.4.11 Starting a UNIX Component via Universal Control . . . 372
1.12.4.12 Stopping a UNIX Component via Universal Control . . . 372
1.12.4.13 Starting an IBM i Component via Universal Control . . . 372
1.12.4.14 Stopping an IBM i Component via Universal Control . . . 373
1.12.5 Maintaining Universal Broker Definitions in UEC Database . . . 374
1.12.5.1 Maintaining Broker Definitions in UEC Database - zOS and Windows . . . 374
1.12.5.2 Maintaining Broker Definitions in UEC Database - zOS . . . 377
1.12.5.3 Maintaining Broker Definitions in UEC Database - Windows . . . 379
1.13 Event Monitoring and File Triggering . . . 380
1.13.1 Event Monitoring and File Triggering - Universal Event Monitor . . . 380
1.13.2 Event Monitoring and File Triggering - UEMLoad . . . 384

1.13.3 Event Monitoring and File Triggering - Examples . . . 386
1.13.3.1 Starting an Event-Driven UEM Server - zOS . . . 387
1.13.3.2 Refreshing an Event-Driven UEM Server - zOS . . . 387
1.13.3.3 Using a Stored Event Handler Record - zOS . . . 388
1.13.3.4 Handling an Event with a Script - zOS . . . 389
1.13.3.5 Handling an Expired Event - zOS . . . 391
1.13.3.6 Continuation Character (-) in zOS Handler Script . . . 392
1.13.3.7 Continuation Character (+) in zOS Handler Script . . . 392
1.13.3.8 Continuation Characters (- and +) in zOS Handler Script . . . 393
1.13.3.9 Using a Stored Event Handler Record - Windows . . . 394
1.13.3.10 Execute Script for Triggered Event Occurrence - Windows . . . 394
1.13.3.11 Handling an Expired Event - Windows . . . 396
1.13.3.12 Add a Single Event Record - Windows . . . 397
1.13.3.13 Add a Single Event Handler Record - Windows . . . 397
1.13.3.14 List All Event Definitions - Windows . . . 398
1.13.3.15 Export Event Definition and Handler Databases - Windows . . . 399
1.13.3.16 List a Single Event Handler Record - Windows . . . 399
1.13.3.17 List Event Definitions and Handlers Using Wildcards - Windows . . . 400
1.13.3.18 Add Record(s) Using Definition File - Windows . . . 400
1.13.3.19 Add Records Remotely Redirected from STDIN - Windows . . . 401
1.13.3.20 Add Records Redirected from STDIN (for zOS) - Windows . . . 401
1.13.3.21 Definition File Format - Windows . . . 402
1.13.3.22 Using a Stored Event Handler Record - UNIX . . . 404
1.13.3.23 Execute Script for Triggered Event Occurrence - UNIX . . . 405
1.13.3.24 Handling an Expired Event - UNIX . . . 406
1.13.3.25 Add a Single Event Record - UNIX . . . 407
1.13.3.26 Add a Single Event Handler Record - UNIX . . . 408
1.13.3.27 List All Event Definitions - UNIX . . . 408
1.13.3.28 List a Single Event Handler Record - UNIX . . . 409
1.13.3.29 Export Event Definition and Handler Databases - UNIX . . . 410
1.13.3.30 List Event Definitions and Handlers Using Wildcards - UNIX . . . 410
1.13.3.31 Add Record(s) Using Definition File - UNIX . . . 411
1.13.3.32 Add Record(s) Remotely Redirected from STDIN - UNIX . . . 411
1.13.3.33 Add Record(s) Remotely Redirected from STDIN (for zOS) - UNIX . . . 412
1.13.3.34 Definition File Format - UNIX . . . 412
1.14 Fault Tolerance Implementation . . . 414
1.14.1 Network Fault Tolerance - Universal Command . . . 415
1.14.2 Network Fault Tolerance - Universal Connector . . . 416
1.14.3 Network Fault Tolerance - Universal Data Mover . . . 416
1.14.4 Manager Fault Tolerance - Universal Command . . . 418
1.14.4.1 Manager Fault Tolerance - Universal Command - Functionality . . . 418
1.14.4.2 Manager Fault Tolerance - Universal Command - Component Management . . . 425
1.14.5 Client Fault Tolerance - Universal Connector . . . 425
1.14.5.1 Client Fault Tolerance - Universal Connector Jobs . . . 426
1.14.5.1.1 Client Fault Tolerance - Universal Connector Jobs - Modes . . . 426
1.14.5.1.2 Client Fault Tolerance - Universal Connector Jobs - Parameters . . . 427
1.14.5.1.3 Client Fault Tolerance - Universal Connector Jobs - Command ID Job Step . . . 428
1.14.5.1.4 Client Fault Tolerance - Universal Connector Jobs - Command Identifier . . . 428
1.14.5.1.5 Client Fault Tolerance - Universal Connector Jobs - Requesting Restart . . . 428
1.14.5.2 Client Fault Tolerance - Universal Connector Process Chains . . . 429
1.14.5.2.1 Client Fault Tolerance - Universal Connector Process Chains - Modes . . . 429
1.14.5.2.2 Client Fault Tolerance - Universal Connector Process Chains - Parameters . . . 430
1.14.5.2.3 Client Fault Tolerance - Universal Connector Process Chains - Dummy Job with Log ID and Command ID Job Steps . . . 431
1.14.5.2.4 Client Fault Tolerance - Universal Connector Process Chains - Command Identifier . . . 431
1.14.5.2.5 Client Fault Tolerance - Universal Connector Process Chains - Requesting Restart . . . 431
1.14.5.3 Sample Command Lines For Working With Client Fault Tolerance . . . 432
1.14.5.3.1 Working With Job Definition Files . . . 432
1.14.5.3.2 Working With Pre-defined SAP Jobs . . . 435
1.14.6 Implementing Fault Tolerance - Examples . . . 438
1.14.6.1 Implementing Manager Fault Tolerance for Windows . . . 438
1.15 Monitoring and Alerting . . . 438
1.15.1 Monitoring and Alerting - Examples . . . 439
1.15.2 Universal Query - zOS . . . 440
1.15.3 Universal Query - UNIX and Windows . . . 440
1.15.4 Universal Query - IBM i . . . 440
1.15.5 Universal Query - Output . . . 441
1.16 Messaging and Auditing . . . 442
1.16.1 Messaging . . . 442
1.16.2 Auditing . . . 444
1.16.3 Creating Write-to-Operator Messages - Examples . . . 445
1.16.3.1 Issue WTO Message to zOS Console . . . 445

1.16.3.2 Issue WTO Message to zOS Console and Wait for Reply . . . 445
1.17 Message Translation . . . 446
1.17.1 Message Translation - Examples . . . 447
1.17.1.1 Translating Error Messages . . . 448
1.17.1.2 Execute Universal Message Translator from zOS . . . 449
1.17.1.3 Execute UMET from zOS Manager (with Table on Remote Server) . . . 450
1.17.1.4 Execute UMET from zOS Manager (with Table on zOS) . . . 451
1.17.1.5 Execute Universal Message Translator from Windows . . . 452
1.17.1.6 Execute Universal Message Translator from UNIX . . . 452
1.17.1.7 Execute Universal Message Translator from IBM i . . . 453
1.18 Network Data Transmission for Universal Agent . . . 453
1.18.1 SSL/TLS Protocol . . . 454
1.18.2 Universal V2 Protocol . . . 457
1.18.3 Universal Application Protocol . . . 458
1.18.4 Network Data Transmission Tuning . . . 459
1.18.5 Network Data Transmission Configurable Options . . . 460
1.19 Event Log Dump for Windows . . . 462
1.19.1 Windows Event Log Dump - Examples . . . 463
1.19.1.1 Execute Universal Event Log Dump from zOS Manager . . . 463
1.19.1.2 Execute Universal Event Log Dump from a Windows Server . . . 463
1.20 z/OS CANCEL Command Support . . . 464
1.20.1 z/OS CANCEL Command Support - Universal Command . . . 464
1.20.2 z/OS CANCEL Command Support - Universal Connector . . . 465
1.20.3 z/OS CANCEL Command Support - Universal Data Mover . . . 465
1.21 UAGRERUN Stand-Alone Execution . . . 466

Universal Agent 7.4.x User Guide

- Overview
- User Guide Information

Currently, IBM i runs Workload Automation 5.1.1.

Information for IBM i in this User Guide refers to this version.

Overview

The Universal Agent 7.3.x User Guide provides information on the enterprise scheduling features of Universal Agent (Universal Command) and the intelligent file transfer features of Universal Agent (Universal Data Mover), and the Universal Agent components that are required as part of the solution presented by each feature.

- Describes how each feature fits into the Universal Agent business solution.
- Illustrates example solutions of how each feature can be implemented.
- Identifies the Agent components used as part of each solution.
- Provides links to technical documentation for the components.

User Guide Information

The Universal Agent user guide contains user and technical information pertaining to both Universal Command and Universal Data Mover, which share many features and functionality.

- Universal Command Overview
- Universal Data Mover Overview
- Universal Agent Features
- Universal Agent Components
- Remote Execution via Universal Command and Universal Data Mover
- Remote Execution for SAP Systems
- Web Services Execution
- Copying Files to and from Remote Systems
- Transferring Files to and from Remote Systems
- Encryption
- Configuration Management for Universal Agent
- Component Management
- Event Monitoring and File Triggering
- Monitoring and Alerting
- Messaging and Auditing
- Message Translation
- Network Data Transmission for Universal Agent
- Event Log Dump for Windows
- z/OS CANCEL Command Support

Universal Command Overview

- Universal Agent
- What is Universal Agent?
  - Workload Types
  - Event-Driven Capabilities
  - Universal Agent
- How Customers use Universal Agent
  - Deployment
- Usage
- Implementation

Universal Agent

Universal Agent (using Universal Command as its core component) is the Universal Automation Center business solution for job scheduling.

Together with Universal Data Mover, Universal Agent forms a common Agent (a single code base to install) that handles both managed file transfers and your enterprise workload automation.

Universal Agent integrates with your current scheduling engine, enabling standardized system-wide processes and procedures.

It allows job execution without regard to the platform or scheduling solution. It also enables the integration of multiple scheduling solutions. You can set up standardized Universal Agent processes to execute any workload anywhere in your environment, allowing job scheduling across platforms without specialized platform-dependent scheduling solutions or training. Using Universal Agent, all elements of the business process are visible. Execution information can be stored in a centralized repository to further reduce the complexities of historical data and audit requirements.

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

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

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

What is Universal Agent?

Universal Agent is a workload automation (job scheduling) agent that can be deployed with any workload automation tool from any vendor on any supported platform. Universal Agent extends workload automation tools to operate across a wide variety of platforms, execute a wide variety of workload types, and provide event-driven automation capabilities.

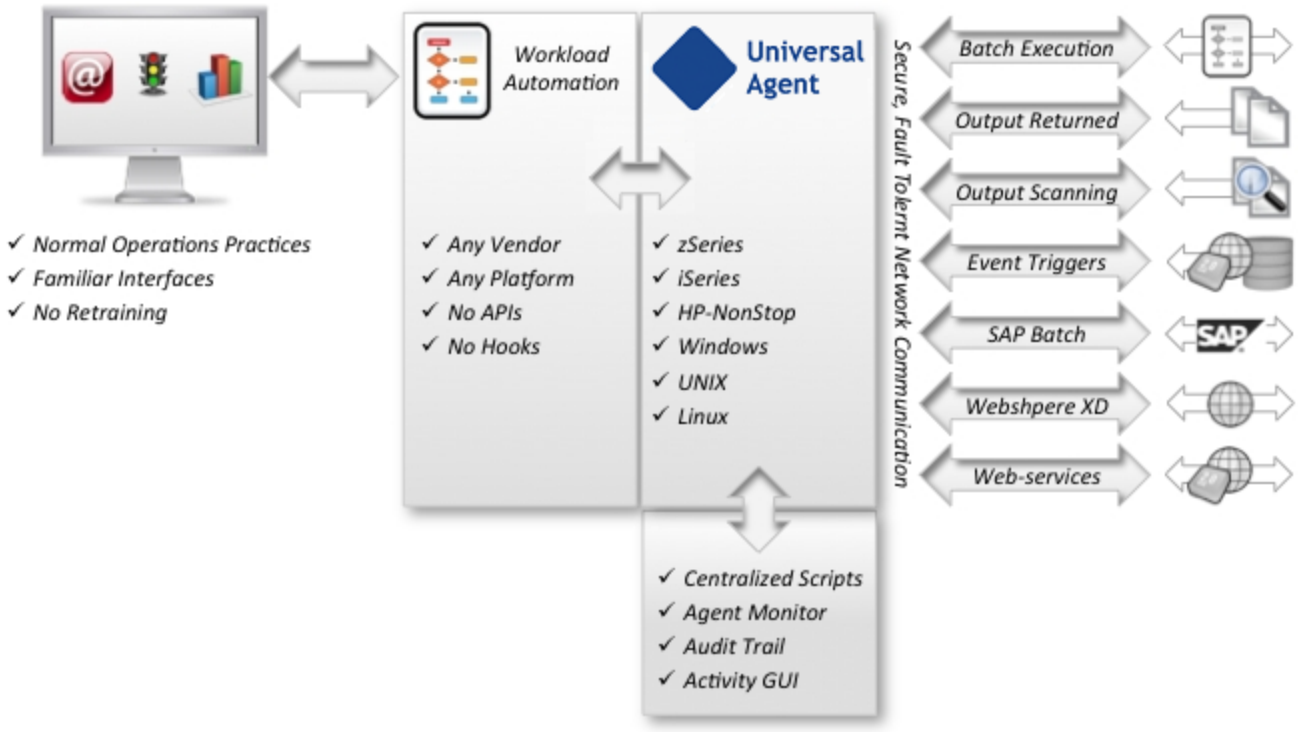
Workload Types

- z/OS batch jobs (Universal Controller only)
- z/OS Started Tasks and Started Jobs (All schedulers)
- z/OS USS commands and scripts (All schedulers)
- z/Linux commands and scripts (All schedulers)
- i5/OS batch jobs and commands (All schedulers)
- Windows commands and scripts (All schedulers)
- UNIX commands and scripts (All schedulers)
- Linux commands and scripts (All schedulers)
- HP NonStop commands and scripts (All schedulers)
- SAP (All schedulers)
- Web services SOAP protocol
- Web services JMS protocol
- Web services MQ Series
- Web services Http(s) protocol
- Websphere XD

Event-Driven Capabilities

- File detection for z/OS systems (Universal Controller only)
- File detection for Windows, UNIX, and Linux Servers
- Web services events for SOAP, JMS, and MQ Series

Universal Agent



How Customers use Universal Agent

Universal Agent makes any job scheduler cross platform and workload automation enabled, provides a lower cost collaborative scheduling agent, and simplifies your infrastructure with a single automation agent for multiple automation tasks.

Deployment

Universal Agent can be deployed in any of the following ways:

- Universal Agent with Universal Controller
- Universal Agent with any non-Stonebranch Scheduler
- Automation Agent for automation tools such as network monitors, enterprise consoles, software distribution, and more.

Usage

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

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

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

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

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

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

Implementation

Universal Agent provides simplified implementation, enabling rapid deployment throughout any environment. A common infrastructure and command language means that deployments are not platform-specific.

User's access to servers and files is managed via native operating system security. Also, user's access to Universal Agent is centrally managed. All Universal Agent installation materials and documentation are delivered electronically via the Stonebranch Customer Portal. This ensures that customers can always access the most current versions and documentation.

All Universal Agent functions and components are delivered in a single install package for each platform. Native operating system packaging simplifies installation. Universal Agent 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 https://www.stonebranch.com/transition-services/migration-implementation-services for more information.

Universal Data Mover Overview

- Universal Data Mover
- Usage
- Universal Data Mover
- Implementation

Universal Data Mover

Universal Data Mover is the Universal Automation Center solution for managed file transfer.

Together with Universal Agent (using Universal Command as its core component), Universal Data Mover forms a common Agent (a single code base to install) that handles both managed file transfers and your enterprise workload automation.

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

Universal Data Mover 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 Universal Data Mover allows you to find information about file transfer activity such as failed transfers and successful transfers, how much data was transferred, and transfer attributes.

Universal Data Mover 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 Universal Data Mover lets you streamline business processes by optimizing the integration of file transfers with your business processes. This helps you avoid delays and maximize revenue.

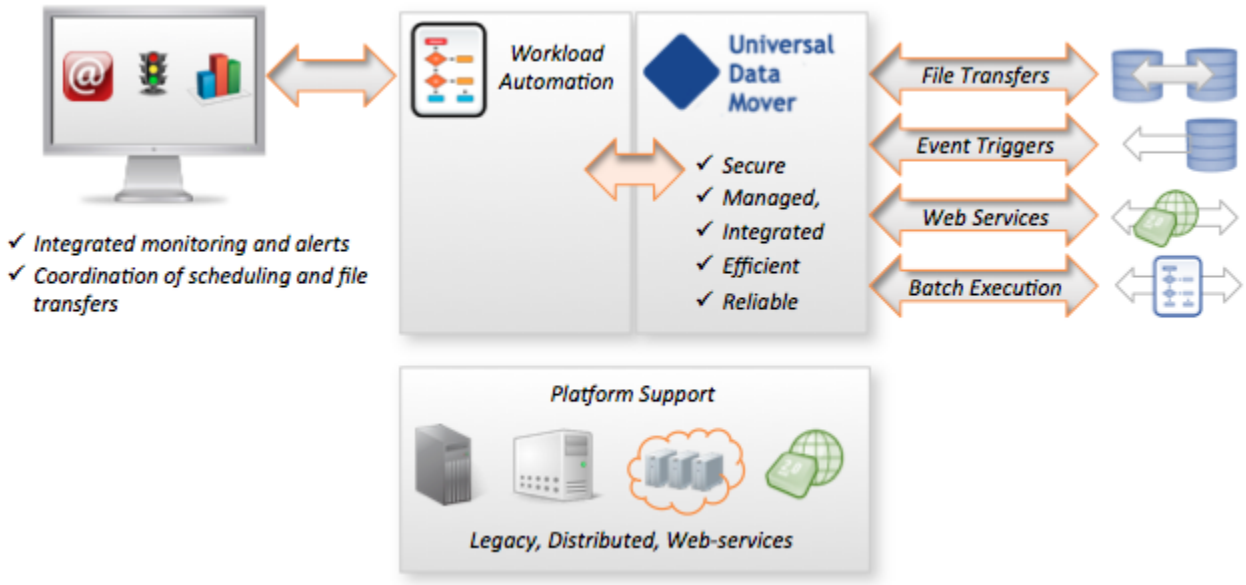
Using Universal Data Mover enables you to securely transfer files to external partners without disruption of their current business processes. The integration capabilities and ease of use provided by Universal Data Mover enable you to manage thousands of servers with minimum interaction.

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

Universal Data Mover 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.

Universal Data Mover delivers flexible visibility tools and capabilities to meet your own business and operational needs for both internal and external communications. With its proactive monitoring, Universal Data Mover 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.

Universal Data Mover



Implementation

Universal Data Mover 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 Universal Data Mover is centrally managed. All Universal Data Mover installation materials and documentation are delivered electronically via the Stonebranch Customer Portal. This ensures that customers can always access the most current versions and documentation.

All Universal Data Mover functions and components are delivered in a single installation package for each platform. Native operating system packaging simplifies installation. Universal Data Mover 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 https://www.stonebranch.com/transition-services/migration-implementation-services for more information.

Universal Agent Features

Features

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

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

The Universal Command component of Universal Agent is a command line interface that allows Remote Execution of all job scheduling to be initiated - regardless of operating system - *from* any machine in your enterprise *to* any machine in your enterprise.

The Universal Data Mover component of Universal Agent allows for the 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.

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

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

For Universal Agent 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.

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

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

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

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

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

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

Universal Agent Components

- Overview
- Universal Command
- Universal Command Agent for SOA
- Universal Data Mover
- Universal Event Monitor
  - UEMLoad
- Universal Event Monitor for SOA
- Universal Enterprise Controller
  - UECLoad
  - Universal Event Subsystem
- Universal Enterprise Controller Client Applications
  - I-Activity Monitor
  - I-Management Console
  - I-Administrator
- Universal Connector for SAP
- Universal Broker
- Universal Automation Center Agent (UAG)
- Universal Message Service (OMS)
- Universal Controller Command Line Interface
- Universal Agent Utilities
  - Universal Certificate
  - Universal Control
  - Universal Copy
  - Universal Database Dump
  - Universal Database Load
  - Universal Display Log File
  - Universal Encrypt
  - Universal Event Log Dump
  - Universal FTP Client
  - Universal Message Translator
  - Universal Install Merge

- Universal Query
- Universal Return Code
- Universal Spool List
- Universal Spool Remove
- Universal Submit Job
- Universal Write-to-Operator
- Additional Documentation
  - Installation and Administration
  - Messages and Codes

Overview

Universal Agent 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 Universal Agent. 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 Command

Universal Command (UCMD), the core component for 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.

The Universal Command interface is operating-system independent. The remote and local systems can be running two different operating systems.

Universal Command consists of two components:

- Manager, on the local system, extends a command line interface to a remote system.
- Server, on the remote system, executes commands on behalf of the manager.

The manager supplies input files to, and receives output files from, the remote command on the server in real-time. As long as the remote command is running, the manager runs. When the remote command ends, the manager ends with the exit status of the remote command. With standard out and standard error as well as the exit status of the remote command available from the manager, there is no need for access to or expertise on the remote operating system.

As such, Universal Command interfaces with your platform-specific job scheduling solutions, providing visibility and control throughout your entire enterprise. This enables you to have an end-to-end view of all workload activity.

Technical Documentation

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

Universal Command 7.4.x Reference Guide

Universal Command 7.4.x Quick Reference Guides

Universal Command Agent for SOA

Universal Command Agent for SOA - the SOA "Publisher" - lets you extend the workload execution and management features of 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. It lets you invoke these workloads using protocols such as JMS, HTTP, and SOAP.

Universal Command Agent for SOA can be initiated from a variety of sources, regardless of platform, such as one or more job scheduling systems, workflow engines, or EAI tools, as well as from business applications and end users, enabling you to consolidate your Internet and message-based workload within your current enterprise scheduling environment.

Universal Command Agent for SOA enables you to:

1. Consolidate your Internet and message-based workload within your current Enterprise Scheduling environment.
2. Use your existing scheduler, or other workload management applications, along with your new or existing Stonebranch components.

3. Use your existing development, test, and production business processes.
4. Use a single point of workload execution that is not tied to specific vendor hardware or software platforms.

> **Note**
> Universal Event Monitor for SOA - the SOA "Listener" - is a file-based event monitoring component available for use with Universal Agent that can be triggered by internet and message-based events.

Technical Documentation

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

Universal Command Agent for SOA 7.4.x Reference Guide

Getting Started with Universal Command Agent for SOA - MQ Connector

Getting Started with Universal Command Agent for SOA - XD Connector

Universal Data Mover

Universal Data Mover (UDM) is the core component for Universal Data Mover'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/TLS 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 7.4.x Reference Guide

Universal Data Mover 7.4.x Quick Reference Guide

Universal Event Monitor

Universal Event Monitor (UEM) 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; that is, 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 Universal Agent 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/TLS 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 7.4.x Reference Guide

Universal Event Monitor 7.4.x Quick Reference Guide

UEMLoad 7.4.x 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 Universal Agent.

Technical Documentation

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

Universal Event Monitor for SOA 7.4.x Reference Guide

Universal Enterprise Controller

Universal Enterprise Controller (UEC) provides alerts for activity and availability of the 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 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 7.3.x Reference Guide

Universal Event Subsystem 7.3.x Event Definitions

UECLoad 7.2.x 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 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 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 Universal Data Mover 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 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 7.4.x User Guide

Universal Connector for SAP

Universal Connector for SAP (USAP) 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.

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

Certified by SAP, Universal Connector uses standard SAP interfaces only, such as XBP3.0, without installing any modules into the SAP environment or onto a SAP server. It installs on a single central platform and connects to any number of SAP systems.

Universal Connector integrates with your output management tools to provide central audit and archive capability for both SAP joblogs and spoollists. Additionally, error messages logged to the SAP system log during the job's execution are copied to its joblog, enabling you to identify and resolve SAP batch issues without requiring access to the SAP system.

Technical Documentation

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

Universal Connector for SAP 7.4.x Reference Guide

Universal Connector for SAP 7.4.x Quick Reference Guide

Universal Broker

Universal Broker (UB), required on all systems running Universal Agent, manages 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 7.4.x Reference Guide

Universal Automation Center Agent (UAG)

Universal Automation Center Agent (UAG) provides agent services for Universal 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 a 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 7.3.x Reference Guide

Universal Message Service (OMS)

Universal Message Service (OMS) is the network communication provider between Universal Controller 7.2.x and Universal Agent 7.2.x.

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 Universal Message Service, see the following document:

Universal Message Service (OMS) 7.3.x Reference Guide

Universal Controller Command Line Interface

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

Technical Documentation

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

Universal Controller Remote Interfaces

Universal Agent Utilities

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

Universal Certificate

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

Universal Control

Universal Control (UCTL) provides the ability to start and stop Universal Agent components, and to refresh component configuration data.

Universal Copy

Universal Copy (UCOPY) provides a means to copy files from either manager-to-server or server-to-manager. (For full-featured managed file transfer, see #Universal Data Mover.)

Universal Database Dump

Universal Database Dump (UDBDUMP), 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 (UDBLOAD), tailored specifically for Stonebranch databases, allows you to restore a database that has been previously dumped.

Universal Display Log File

Universal Display Log File (UDSPLOGF) 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 (UENCRYPT) encrypts the contents of command files into an unintelligible format (for privacy reasons).

Although all 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 FTP Client

Universal FTP Client (UFTP) transfers files to and from servers using any of the following file transfer protocols: FTP, FTPS, SFTP, and TFTP.

UFTP on a local machine (localhost) communicates with FTP Server software on a remote host and transfers files specified on its command line to another file on the client/server.

Files can be listed using a comma as a delimiter (a comma-delimited list of files). The transfer result is never concatenated in a single file; each file copies to a file of the same name.

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 a 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 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 Universal Agent installation to query any Universal Broker in the Stonebranch infrastructure.

Universal Return Code

Universal Return Code (URC) 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 (USLIST) 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 (USLRM) 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 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 Universal Agent Utilities, see the following documents:

Universal Agent Utilities 7.3.x Reference Guide

Universal Certificate 7.3.x Quick Reference Guide

Universal Control 7.2.x Quick Reference Guide

Universal Query 7.2.x Quick Reference Guide

Additional Documentation

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

Installation and Administration

Universal Agent 7.3.x Installation, Upgrade, and Applying Maintenance

Universal Agent 7.3.x Administration

Universal Agent 7.3.x Installation Requirements and Summary

Universal Agent 7.3.x Installation Quick Start Guides

Messages and Codes

Universal Agent 7.3.x Messages and Codes

Remote Execution via Universal Command and Universal Data Mover

The information on Remote Execution is provided on the following pages:

- Universal Command - Remote Execution
  - Remote Execution via Universal Command - Primer
  - Remote Execution via Universal Command - Examples
- Universal Data Mover - Remote Execution
  - Remote Execution via Universal Data Mover - Primer
  - Remote Execution via Universal Data Mover - Examples

Universal Command - Remote Execution

- Overview
- Remote Execution Components
- Additional Information

Overview

This page provides information on the Remote Execution feature of Universal Agent.

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

Remote Execution Components

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

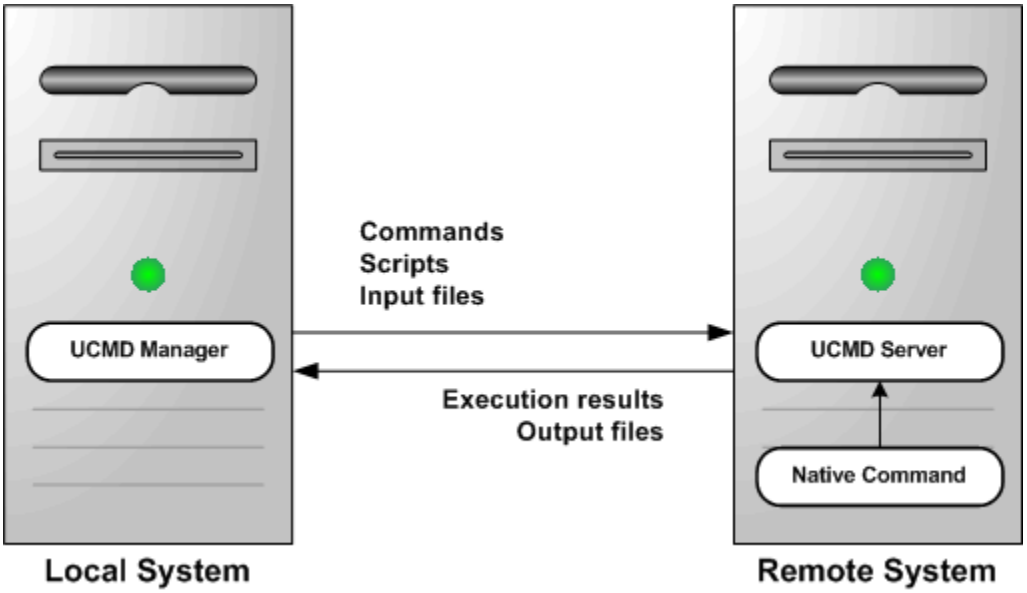
1. Universal Command Manager runs on the local system. The Manager initiates the work on the remote system.
2. Universal Command Server runs on the remote systems. It executes work on behalf of a Universal Command Manager.

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

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

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

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



Additional Information

The following pages provide additional detailed information for Remote Execution:

- Remote Execution via Universal Command - Primer
- Remote Execution via Universal Command - Examples

Remote Execution via Universal Command - Primer

- Overview
- Remote Execution Primer Examples
- Executing Universal Command Manager on z/OS
  - Components
- Executing Universal Command Manager on Windows
  - Components
- Executing Universal Command Manager on UNIX
  - Components
- Executing Universal Command Manager on IBM i
  - Components
- Executing Universal Command Manager on HP NonStop
  - Components

Overview

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

Prior to reading this page, read the Overview of Remote execution, as this page builds upon the material presented in the Overview. The primer discussions are from the perspective of the initiating system where the Universal Command (UCMD) Manager component is executed.

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

The primer examples demonstrate how to execute a command on a remote system using the Universal Command Manager component. All examples use the same set of configuration options (identified in the table below). The actual option names can be different, depending on the operating system on which the UCMD Manager executes. This difference is due to operating system conventions or standards that UCMD abides by.

Remote Execution Primer Examples

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

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

Executing Universal Command Manager on z/OS

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

A UCMD Manager JCL procedure is provided with the Universal Agent installation to simplify JCL requirements. The JCL procedure name is **UCMDPRC**; it is located in the **SUNVSAMP** product library. See the Universal Command 7.3.x Reference Guide for more information on the UCMDPRC procedure.

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

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

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

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

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

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

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

Executing Universal Command Manager on Windows

Universal Command Manager is run as a command on Windows.

The following command and command line options execute UCMD Manager.

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

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

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

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

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

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

Components

Universal Command Manager for Windows

Executing Universal Command Manager on UNIX

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

The following command and command line options execute UCMD Manager.

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

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

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

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

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

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

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

Components

Universal Command Manager for UNIX

Executing Universal Command Manager on IBM i

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

The following CL command and parameters execute UCMD Manager.

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

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

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

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

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

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

Components

Universal Command Manager for IBM i

Executing Universal Command Manager on HP NonStop

Universal Command Manager is run as a TACL command.

The following command and command line options execute UCMD Manager.

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

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

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

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

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

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

Components

Universal Command Manager for HP NonStop

Remote Execution via Universal Command - Examples

- Introduction
- Remote Execution Examples - z/OS
- Remote Execution Examples - Windows
- Remote Execution Examples - UNIX
- Remote Execution Examples - IBM i

Introduction

The Remote Execution examples illustrated in these pages are specific to the operating systems supported by Universal Agent.

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

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

Remote Execution Examples - z/OS

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

Remote Execution Examples - Windows

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

Remote Execution Examples - UNIX

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

Remote Execution Examples - IBM i

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

> **Note**

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

Back up UNIX Directory to zOS Dataset

- Back up UNIX Directory to z/OS Dataset
  - SYSIN Options
  - Components

Back up UNIX Directory to z/OS Dataset

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

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

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

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

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-host	Host name or IP address of the remote system on which to execute the script.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-script	ddname from which to read the script file. The script file is sent to the remote system by the UCMD Manager for execution.
-stdout	Starts the stdout options. All options read afterwards are applied to the stdout file. The first option not recognized as a standard file option terminates the stdout option list.

-mode	Transfer mode for the stdout file: **binary**. Since a backup file is a tar, compressed format contains binary data, it should not be translated as text data.

Components

Universal Command Manager for z/OS

Restore UNIX Directory Backup from zOS Dataset to UNIX Directory

- Restore UNIX Directory Backup from z/OS Dataset to UNIX Directory
  - SYSIN Options
  - Components

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

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

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

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

tar -xzvBf - || exit 8

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

SYSIN Options

The SYSIN options used in this example are:

Option	Description

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

Components

Universal Command Manager for z/OS

Directory Listing for UNIX Server from zOS

- Directory Listing for UNIX Server from z/OS
  - SYSIN Options
  - Components

Directory Listing for UNIX Server from z/OS

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

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

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

Universal Command Manager for z/OS

Directory Listing for Windows Server from zOS

- Directory Listing for Windows Server from z/OS
  - SYSIN Options
  - Components

Directory Listing for Windows Server from z/OS

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

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

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

Universal Command Manager for z/OS

Provide Network Status of Remote UNIX from zOS

- Provide Network Status of Remote UNIX from z/OS
  - SYSIN Options
  - Components

Provide Network Status of Remote UNIX from z/OS

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

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

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

Universal Command Manager for z/OS

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

- Use UNIX tee Command to Store stdout to Local Server and z/OS
  - SYSIN Options
  - Components

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

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

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

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

```
-pwd      password
/*
```

SYSIN Options

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

Components

Universal Command Manager for z/OS

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

- Use an Encrypted Command File for User ID and Password on z/OS
  - Create an Encrypted Command File
    - UNVIN Options
  - SYSIN Options
  - Components

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

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

Create an Encrypted Command File

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

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

UNVIN Options

The UNVIN options used in this example are:

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

Use an Encrypted Command File

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

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

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

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

Universal Command Manager for z/OS

Override Standard zOS IO File ddnames

- Override Standard z/OS I/O File ddnames
  - SYSIN Options
  - Components

Override Standard z/OS I/O File ddnames

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

The example:

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

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

Universal Command Manager for z/OS

Override zOS Standard Files with Procedure Symbolic Parameters

- Override z/OS Standard Files with Procedure Symbolic Parameters
  - SYSIN Options
  - Components

Override z/OS Standard Files with Procedure Symbolic Parameters

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

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

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

```
-pwd      password
-cmd      command
/*
```

SYSIN Options

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

Components

Universal Command Manager for z/OS

Specifying UCMD for zOS Options with the EXEC PARM

- Specifying UCMD for z/OS Options with the EXEC PARM
  - SYSIN Options
  - Components

Specifying UCMD for z/OS Options with the EXEC PARM

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

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

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

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

SYSIN Options

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

Components

Universal Command Manager for z/OS

Executing an Existing Windows .bat File from zOS

- Execute an Existing Windows .bat file from z/OS
  - SYSIN Options
  - Components

Execute an Existing Windows .bat file from z/OS

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

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

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

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

Universal Command Manager for z/OS

Using Manager Fault Tolerance from zOS

- Using Manager Fault Tolerance from z/OS
  - SYSIN Options
  - Components

Using Manager Fault Tolerance from z/OS

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

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

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

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

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

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

Universal Command Manager for z/OS

Restarting a Manager Fault Tolerant UCMD Manager on zOS

- Restarting a Manager Fault Tolerant UCMD Manager on z/OS
  - SYSIN Options
  - Components

Restarting a Manager Fault Tolerant UCMD Manager on z/OS

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

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

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

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

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

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

SYSIN Options

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

Components

Universal Command Manager for z/OS

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

- Automatically Create a Unique z/OS Command ID Using CA-Driver Variables
  - Example 1
  - Example 2
  - SYSIN Options
  - Components

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

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

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

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

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

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

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

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

(See the following examples.)

Example 1

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

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

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

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

Expanded Results:

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

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

Example 2

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

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

Expanded Results:

//S1    EXEC UCMDPRC
//SYSIN DD *
```

```
-cmdid unixpayrolljob1PRD000010030001032625
-cmd command -host hostname -userid username -pwd password
-managerft yes -restart auto
/*
```

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

SYSIN Options

The SYSIN options used in these examples are:

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

Components

Universal Command Manager for z/OS

Automatically Create a Unique zOS Command ID Using Zeke Variables

- Automatically Create a Unique z/OS Command ID for your Universal Agent Process Using Zeke Variables
  - SYSIN Options
  - Components

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

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

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

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

Expanded Results:
```

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

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

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-cmd	Command to execute on the remote system.
-script	ddname from which to read a script file. The script file is sent to the remote system by UCMD Manager for execution.
-host	Host name or IP address of the remote system on which to execute the script.
-encryptedfile	ddname from which to read an encrypted command file created with the Universal Agent Universal Encrypt (UENCRYPT) utility.
-managerft	Specification for whether or not the Manager Fault Tolerance (MFT) feature is used. A value of **yes** specifies that MFT should be used.
-cmdid	Unique command ID associated with the remote unit of work.
-restart	Specification for whether or not the manager is requesting restart. A value of **yes** indicates the Manager is being restarted.

Components

Universal Command Manager for z/OS

Automatically Create a Unique zOS Command ID Using OPC Variables

- Automatically Create a Unique z/OS Command ID for your Universal Agent Process Using OPC Variables
  - SYSIN Options
  - Components

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

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

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

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

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

```
//S1 EXEC UCMDPRC
//LOGONDD  DD DISP=SHR,DSN=hlq.encrypted.file
//SCRIPTDD DD *
DIR
SYSIN    DD *
-cmdid payrolljob&CYMD.
```

```
&CHHMMSSX.                                                    <== OPC
variables
-script scriptdd -host dallas -encryptedfile logondd -managerft yes -restart
no
/*

Expanded Results:

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

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

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-cmd	Command to execute on the remote system.
-script	ddname from which to read a script file. The script file is sent to the remote system by UCMD Manager for execution.
-host	Host name or IP address of the remote system on which to execute the script.
-encryptedfile	ddname from which to read an encrypted command file created with the Universal Agent Universal Encrypt (UENCRYPT) utility.
-managerft	Specification for whether or not the Manager Fault Tolerance (MFT) feature is used. A value of **yes** specifies that MFT should be used.
-restart	Specification for whether or not the manager is requesting restart. A value of **yes** indicates the Manager is being restarted.

Components

Universal Command Manager for z/OS

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

- Universal Submit Job from z/OS to IBM i using the Remote Reply Facility
  - USBMJOB Options
  - SYSIN Options
  - Components

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

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

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

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

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

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

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

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

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

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

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

USBMJOB Options

The USBMJOB options used in this example are:

Option	Description
CMD	Specifies a command that runs in the submitted batch job. The command can be a maximum of 3000 characters.
RMTRPY	Specifies whether USBMJOB will use the Remote Reply Facility.
RMTREFRESH	Specifies the number of seconds to refresh the z/OS console message if no reply is received. The previous message is deleted from the console and a new one is issued.
RMTMSGPRFX	Specifies a text string up to 12 characters that will prefix the message written to the z/OS console.
RMTHOST	Specifies the host name or IP address of the z/OS system on which the reply message is issued.
MSGCMDPATH	Specifies the path name of the z/OS USS **uwto** program. If the **uwto** program is in the system PATH, this parameter is not required.
RMTUSER	Specifies the z/OS user ID with which the **uwto** program is executed.
RMTPWD	Specifies the password for the z/OS user ID specified with the RMTUSR parameter.

SYSIN Options

The SYSIN options used in this example are:

Option	Description

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

Components

Universal Command Manager for z/OS

Universal Submit Job

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

- Executing Universal Return Code within a Script via Universal Command Manager for z/OS
  - SYSIN Options
  - Components

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

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

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

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

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

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

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

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

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

SYSIN Options

The SYSIN options used in this example are:

Option	Description

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

Components

Universal Command Manager for z/OS

Universal Return Code

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

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

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

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

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

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

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

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

Script Options

The script options used in this example are:

Option	Description

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

Universal Command Manager for z/OS

Universal Return Code

Universal Message Translator

Back up UNIX Directory to Windows

- Back up UNIX Directory to Windows
  - Command Line Options
  - Components

Back up UNIX Directory to Windows

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

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

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

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

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

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

Command Line Options

The command line options used in this example are:

Option	Description
-script	File name of a script file. The script file is sent to the remote system for execution.

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

Components

Universal Command Manager for Windows

Restore UNIX Directory Backup from Windows to UNIX

- Restore UNIX Directory Backup from Windows to UNIX
  - Command Line Options
  - Components

Restore UNIX Directory Backup from Windows to UNIX

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

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

The following figure illustrates the script to perform the restore.

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

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

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

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

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

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

Command Line Options

The command line options used in this example are:

	Description

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

Components

Universal Command Manager for Windows

Provide Network Status of Remote UNIX from Windows

- Provide Network Status of Remote UNIX from Windows
  - Command Line Options
  - Components

Provide Network Status of Remote UNIX from Windows

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

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

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

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

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

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

The report is written to the standard out of UCMD.

Command Line Options

The command line options used in this example are:

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

Components

Universal Command Manager for Windows

Redirect Standard Out and Standard Error to Windows

- Redirect Standard Out and Standard Error to Windows
  - Command Line Options
  - Components

Redirect Standard Out and Standard Error to Windows

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

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

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

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

Command Line Options

The command line options used in this example are:

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

Components

Universal Command Manager for Windows

Start UNIX Background Process from Windows

- Start UNIX Background Process from Windows
  - Command Line Options
  - Components

Start UNIX Background Process from Windows

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

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

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

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

Command Line Options

The command line options used in this example are:

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

Components

Universal Command Manager for Windows

Redirect Standard Input from Initiating System on Windows

- Redirect Standard Input from Initiating System on Windows
  - Command Line Options
  - Components

Redirect Standard Input from Initiating System on Windows

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

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

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

Command Line Options

The command line options used in this example are:

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

Components

Universal Command Manager for Windows

Universal Submit Job from Windows to IBM i

- Universal Submit Job from Windows to IBM i
  - SYSIN Options
  - Components

Universal Submit Job from Windows to IBM i

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

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

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

Universal Command Manager for Windows

Universal Submit Job

Provide Network Status of Remote Windows from UNIX

- Provide Network Status of Remote Windows from UNIX
  - Command Line Options
  - Components

Provide Network Status of Remote Windows from UNIX

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

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

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

```
echo
-------------------------------------------------------------------------------
-----------
```

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

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

The report is written to the standard out of UCMD.

Command Line Options

The command line options used in this example are:

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

Components

Universal Command Manager for UNIX

Redirect Standard Out and Standard Error to UNIX

- Redirect Standard Out and Standard Error to UNIX
  - Command Line Options
  - Components

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

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

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

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

Command Line Options

The command line options used in this example are:

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

Components

Redirect Standard Input from Initiating System to UNIX

- Redirect Standard Input from Initiating System to UNIX
  - Command Line Options
  - Components

Redirect Standard Input from Initiating System to UNIX

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

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

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

Command Line Options

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

Components

Universal Command Manager for UNIX

Redirect Standard Input in UNIX Background Process

- Redirect Standard Input in UNIX Background Process
  - Command Line Options
  - Components

Redirect Standard Input in UNIX Background Process

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

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

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

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

Command Line Options

The command line options used in this example are:

Option	Description
-script	File from which to read a script file. The script file is sent to the remote system for execution.

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

Components

Universal Command Manager for Windows

Issue Universal Submit Job from UNIX to IBM i

- Issue Universal Submit Job from UNIX to IBM i
  - SYSIN Options
  - Components

Issue Universal Submit Job from UNIX to IBM i

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

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

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

Universal Command Manager for UNIX

Universal Submit Job

Provide Network Status of Remote Windows from IBM i

- Provide Network Status of Remote Windows from IBM i
  - Command Line Options
  - Components

Provide Network Status of Remote Windows from IBM i

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

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

```
echo System Status
echo
------------------------------------------------------------------------------
-----------
date /t
```

```
time /t
echo
------------------------------------------------------------------------------
-----------
netstat -se
echo
------------------------------------------------------------------------------
-----------
netstat -a
echo
------------------------------------------------------------------------------
-----------
```

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

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

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

Command Line Options

The command line options used in this example are:

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

Components

Universal Command Manager for IBM i

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

- Execute Script to Provide Network Status of Remote Windows from IBM i
  - Command Line Options
  - Components

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

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

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

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

Command Line Options

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

PWD	Password for the user ID.

Components

Universal Command Manager for IBM i

Display Library with Manager Fault Tolerance Active Using USBMJOB

- Display Library with Manager Fault Tolerance Active Using USBMJOB
  - Command Line Options
  - Components

Display Library with Manager Fault Tolerance Active Using USBMJOB

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

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

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

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

USBMJOB outputs data via the standard output file stream (stdout) and outputs job logs and error messages via the standard error file stream (stderr). The system takes data sent back to UCMD and stores it in **out400.txt**; it takes any error messages and the job logs and stores them in **er r400.txt**.

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

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

Command Line Options

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

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

Components

Universal Command Manager for IBM i

Universal Submit Job from zOS to IBM i

- Universal Submit Job from z/OS to IBM i
  - SYSIN Options
  - USBMJOB Options
  - Components

Universal Submit Job from z/OS to IBM i

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

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

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

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

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

SYSIN Options

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

USBMJOB Options

The USBMJOB option used in this example is:

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

Components

Universal Command Manager for IBM i

Universal Submit Job

Universal Data Mover - 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 Universal Agent is used to execute work on the remote system.

These pages provide information on the remote execution features and functionality of the Universal Data Mover business solution.

Universal Data Mover 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

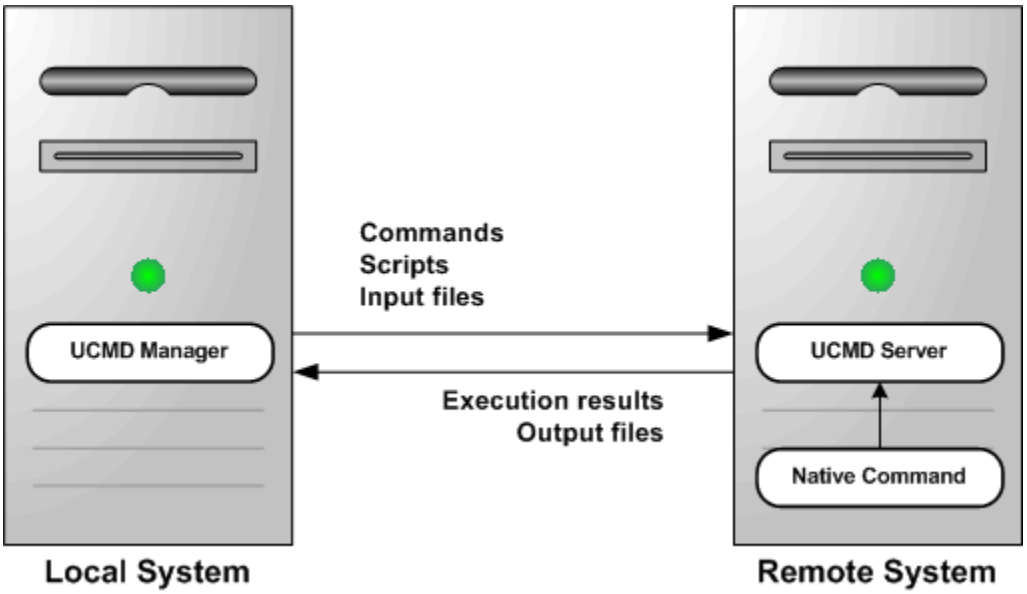
Universal Data Mover Remote Execution using Universal Command consists primarily of two Universal Agent components:

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 Universal Data Mover - Primer

- Overview
- Remote Execution Examples - exec Command Parameters
- Remote Execution Requirements
  - Executing the Examples
- Remote Execution Requirements for z/OS
- Remote Execution Requirements for Windows
- Remote Execution Requirements for UNIX
- Remote Execution Requirements for IBM i

Overview

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

> **Note**
> Please read Universal Data Mover - Remote Execution 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 Universal Data Mover 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 7.3.x 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 7.3.x 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 Universal Agent configuration.

For information on securing access to Universal Agent components, see Universal Agent 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 Universal Data Mover - 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
  - UDM exec Command Parameters
  - Components

Windows Directory Listing Using a Batch File - Default Directory

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

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

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

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

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

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

Output sent to **stdout.txt**.

```
C:\Program Files\Universal\UCmdHome\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
  - UDM exec Command Parameters
  - Components

Windows Directory Listing Using a Batch File - Returned File

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

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

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

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

1. Echo is turned off to minimize the amount of information in the transaction log due to its size. You are encouraged to set up the example and work through the transaction log.
2. Set a variable, **outdir**, for later use. Instead of setting the variable inside of the UDM script, the variable and its associated value could have been provided externally via a script option.
3. Open the UDM 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
  - UDM Script Explanation
  - UDM exec Command Parameters
  - Components

UNIX Listing Using a Shell Script

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

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

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

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

UDM Script Explanation

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

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

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

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

Output sent to **stdout.txt**.

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

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

```
2011.07.28 10.13.06.845 UNV2800I Universal Data Mover 6.3.0 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 6.3.0 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
  - UDM Script Explanation
  - UDM exec Command Parameters
  - Components

UNIX - Integrating UDM with FTP Using a Shell Script

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

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

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

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

```
1. set echo=yes
2. open rmt=192.168.20.47 user=joe pwd=abcdefg port=7887
```

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

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. The open command assigns the logical name 'rmt' to the remote system at IP address 192.168.20.47 and assigns the logical name **local** to the system on which the UDM Manager is running. Setting up **lo cal** is done automatically by UDM for two-party transfers when a second system is not specified.
3. Change the transfer mode from binary to text.
4. Change the Creation Option attribute for the local server from **new** to **replace**. Without this change, existing files cannot be replaced with new copies.
5. Change the current directory on the remote Windows system to **C:\tmp\tmp**. This is the directory from which the file is pulled.
6. Change the current directory on the local UNIX system to **/home/joe/wrk/xmp/dmzFtp**. This is the directory into which the file is pulled.
7. Use the UDM copy command to transfer the file from the remote Windows system to the local UNIX system.
8. Execute the shell script on the local system to FTP the file to the 2nd system inside the firewall. In this example, the exec command uses the UCMD server running on the local system to execute the shell script just as if it resided on a remote system. The port must be specified on the command if it is set to a value other than the default value.
9. Execute the **ls** command on the remote system to show that the file was copied. In a production environment, a process could be started to do something productive with the transferred file.
10. Quit command stops UDM script execution and the UDM script completes.

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
  - UDM Script Explanation
  - UDM exec Command Parameters

UNIX - Integrating UDM with FTP Using a Command Reference

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

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

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

UDM Script Explanation

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

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

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

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

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

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


ftp -ipnv $1 <
```

UDM exec Command Parameters

The exec command parameters used in this example are:

Parameter	Description
cmdref	Command Reference file name and, optionally, options to be passed to the command or script.
user	Remote user ID with which to authenticate and execute the command on the remote system.
pwd	Password with which to authenticate the user ID on the remote system.
port	Port that the Universal Broker is listening on for the remote machine. The port is inherited from the UDM Manager's configuration file unless explicitly overridden in the call to the 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
  - UDM Script Explanation
  - Operating System-Specific Information

- UDM exec Command Parameters
- Components

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

This example demonstrates using the _execrc built-in variable.

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

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

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

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

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

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

UDM Script Explanation

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

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

Operating System-Specific Information

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

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

Remote Execution for SAP Systems

- Overview
  - Work Requests
- Detailed Information

Overview

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

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

The Universal Connector for SAP component of Universal Agent is used to execute this work on the remote SAP system.

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

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

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

Work Requests

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

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

- Retrieve output device information

Detailed Information

The following pages provide detailed information for Remote Execution for SAP Systems:

- Mass Activities Support in Universal Connector
- Mass Activities Support Example for zOS
- Batch Input Monitoring in Universal Connector
- Batch Input Monitoring Example for zOS
- Remote Execution for SAP Systems - Examples

Remote Execution for SAP Systems - Examples

The following Remote Execution for SAP are specific to the operating systems supported by Universal Agent. The examples demonstrate the use of Universal Connector to define SAP jobs.

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

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

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

Remote Execution for SAP Systems Examples - z/OS

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

Remote Execution for SAP Systems Examples - UNIX

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

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

- Define Job, Run Job, Get Output, and Purge Job
  - Command Options
  - Components

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

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

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

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

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

Command Options

The command options used are:

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

Components

Universal Connector for z/OS

Universal Connector for SAP for UNIX

Submitting Job to SAP Using SAP Job as Template - zOS

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

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

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

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

```
   -pwd       sappwd
   -sub
   -jobname   USPSUB1
   -jobid     12345678
/*
```

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

SYSIN Options

SYSIN options used in this example are:

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

Components

Universal Connector for z/OS

Submitting Job to SAP Using Job Definition File - zOS

- Submitting a Job to an SAP System Using a Universal Connector Job Definition File - z/OS
  - SYSIN Options
  - Components

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

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

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

 /* ABAP Step statement. */
 ABAP_STEP            = "STEP 1"
    ABAP_PROGRAM_NAME = "BTCSPOOL";
//SYSIN    DD  *
   -dest   CF5
   -client 800
```

```
   -userid sapuid
   -pwd    sappwd
   -sub    JOBDEF
/*
```

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

SYSIN Options

SYSIN options used in this example are:

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

Components

Universal Connector for z/OS

Running Job on SAP Using SAP Job - zOS

- Running a Job on an SAP System Using a Pre-existing SAP Job - z/OS
- SYSIN Options
  - Components

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

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

```
//USPRUN1  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*****************************************************************
//* Description
//* -----------
//* This sample will:
//* 1. Submit a new job to an SAP system using a pre-existing SAP
//*    job as a template.
//* 2. Start the newly created job.
//* 3. Wait for the job to complete.
//* 4. Return the job log.
//* 5. Return the spool list.
//* 6. The SAP job completion status will be mapped to an exit
//*    code.  USAP will exit with the mapped exit code.
//*
//* NOTE: This job assumes (and requires) that a job already
//* exists on the SAP system with:
//* Job Name: USPRUN1
//* Job ID:   12345678
//*
//         JCLLIB ORDER=#SHLQ.UNV.SUNVSAMP
//*
```

```
//STEP1    EXEC USPPRC
//SYSIN    DD  *
   -dest      CF5
   -client    800
   -userid    sapuid
   -pwd       sappwd
   -sub
   -jobname   USPRUN1
   -jobid     12345678
   -start
   -wait
   -joblog    yes
   -spoollist yes
/*
```

SYSIN Options

SYSIN options used in this example are:

Command Options	Description
-dest	Named set of connection parameters (destination) 'CF5'. These connection parameters are used for communications with the SAP system. The default file for destination parameters is **#HLQ.UNV.USPRFC00**.
-client	SAP client number that the Universal Connector will communicate with.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-sub	Specification that Universal Connector will issue the SUBMIT command.
-jobname	Job name of the SAP job that will be used as a template.
-jobid	Job ID of the SAP job that will be used as a template.
-start	Specification that Universal Connector will instruct the SAP system to start the submitted job.
-wait	Specification that Universal Connector will monitor the started job until it completes.
-joblog	Specification that Universal Connector will return the SAP log for the started job.
-spoollist	Specification that Universal Connector will return any spool lists created by the started job.

Components

Universal Connector for z/OS

Running Job on SAP Using Job Definition File - zOS

- Running a Job on an SAP System Using a Universal Connector Job Definition File - z/OS
  - SYSIN Options
  - Components

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

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

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

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

SYSIN Options

SYSIN options used in this example are:

Command Options	Description
-dest	Named set of connection parameters (destination) 'CF5'. These connection parameters are used for communications with the SAP system. The default file for destination parameters is **#HLQ.UNV.USPRFC00**.
-client	SAP client number that Universal Connector will communicate with.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-sub	Specification that Universal Connector will issue the SUBMIT command.
-start	Specification that Universal Connector will instruct the SAP system to start the submitted job.
-wait	Specification that Universal Connector will monitor the started job until it completes.
-joblog	Specification that Universal Connector will return the SAP log for the started job.
-spoollist	Specification that Universal Connector will return any spool lists created by the started job.

Components

- Running an SAP Job on a Specific SAP Server - z/OS
  - SYSIN Options
  - Components

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

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

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

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

SYSIN Options

SYSIN options used in this example are:

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

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

Components

Universal Connector for z/OS

Variant Substitution - zOS

- Variant Substitution - z/OS
  - SYSIN Options
  - Components

Variant Substitution - z/OS

This example demonstrates the use of variant substitution.

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

```
//USPVARSB  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*****************************************************************
//* Description
//* -----------
//* This sample demonstrates the use of USAP's target_variant
//* option to perform variant substitution when using pre-defined
//* SAP jobs as templates.
//*
//* NOTE: This job assumes (and requires) that a job already
//* exists on an the SAP system with:
//* Job Name: VARSBST1
//* Job ID:   12345678
//*
//* This sample will:
//* 1. Modify variants SBT1 and SBT2 with values required for this
//*    job run.
//* 2. Submit a new job to the SAP system using a pre-existing SAP
//*    job as a template.
//* 3. Perform variant substitution on the newly created job.  The
//*    newly created job will now use variants SBT1 and SBT2 for
//*    steps 1 and 2 respectively (regardless of what variants
//*    were defined in the template job).
//* 4. Wait for the job to complete.
//* 5. Return the job log.
//* 6. Return the spool list.
//*
```

```
//         JCLLIB ORDER=#SHLQ.UNV.SUNVSAMP
//*
//*****************************************************************
//* Modify variant 'SBT1' for ABAP program RSUSR002
//*****************************************************************
//STEP1    EXEC USPPRC
//VARDEF   DD  *
 /* Variant Header statement. */
 VARIANT_NAME   = "SBT1"
    REPORT      = "RSUSR002";
 /* User */
 SELNAME        = "USER"
    KIND        = "S"
    SIGN        = "I"
    OPTION      = "CP"
    LOW         = "STONEBRANCH"
    HIGH        = "";
//SYSIN    DD  *
   -dest         CF5
   -client       800
   -userid       sapuid
   -pwd          sappwd
   -modify       VARDEF
/*
//*****************************************************************
//* Modify variant 'SBT2' for ABAP program RSUSR002
//*****************************************************************
//STEP2    EXEC USPPRC
//VARDEF   DD  *
 /* Variant Header statement. */
 VARIANT_NAME   = "SBT2"
    REPORT      = "RSUSR002";
 /* User */
 SELNAME        = "USER"
    KIND        = "S"
    SIGN        = "I"
    OPTION      = "CP"
    LOW         = "STONEBRANCH1"
    HIGH        = "";
//SYSIN    DD  *
   -dest         CF5
   -client       800
   -userid       sapuid
   -pwd          sappwd
   -modify       VARDEF
/*
//*****************************************************************
//* Run SAP job using a pre-defined SAP job as a template and
//* perform variant substitution.
//*
//* NOTE: This job assumes (and requires) that a job already
//* exists on an the SAP system with:
//* Job Name: VARSBST1
//* Job ID:   12345678
//*
//* The pre-defined job must have ABAP program RSUSR002 defined in
//* step one and step two.
```

```
//*************************************************************
//STEP3    EXEC USPPRC
//SYSIN    DD  *
   -dest          CF5
   -client        800
   -userid        sapuid
   -pwd           sappwd
   -run
   -jobname       VARSBST1
   -jobid         12345678
   -target_variant 1,SBT1;2,SBT2
/*
```

SYSIN Options

SYSIN options used in this example are:

Command Options	Description
-dest	Named set of connection parameters (destination) 'CF5'. These connection parameters are used for communications with the SAP system. The default file for destination parameters is **#HLQ.UNV.USPRFC00**.
-client	SAP client number that Universal Connector will communicate with.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-sub	Specification that Universal Connector will issue the SUBMIT command.
-start	Specification that Universal Connector will instruct the SAP system to start the submitted job.
-targetserver	Target server for the SAP job to run on.
-wait	Specification that Universal Connector will monitor the started job until it completes.
-joblog	Specification that Universal Connector will return the SAP log for the started job.
-spoollist	Specification that Universal Connector will return any spool lists created by the started job.

Components

Universal Connector for z/OS

Creating a Variant Substitution Using GENERATE VARDEF Command - zOS

- Creating a Universal Connector Variant Definition Using the GENERATE VARDEF Command - z/OS
  - SYSIN Options
  - Components

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

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

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

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

```
//USPGEN1  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*************************************************************
//* Description
//* -----------
```

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

SYSIN Options

SYSIN options used in this example are:

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

Components

Universal Connector for z/OS

Creating a Job Definition Using GENERATE JOBDEF Command - zOS

- Creating a Universal Connector Job Definition Using the GENERATE JOBDEF Command - z/OS
  - SYSIN Options
  - Components

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

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

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

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

```
//USPGEN2  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*****************************************************************
//* Description
//* -----------
```

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

SYSIN Options

SYSIN options used in this example are:

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

Components

Universal Connector for z/OS

Submitting an SAP Job Using SAP Job as Template - UNIX

- Submitting a Job to an SAP System Using a Pre-existing SAP Job as a Template - UNIX
  - Command Line Options
  - Components

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

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

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

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

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

The following figure illustrates the command to submit the job.

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

Command Line Options

Command line options used in this example are:

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

Components

Universal Connector for SAP for UNIX

Submitting an SAP Job Using Job Definition File - UNIX

- Submitting a Job to an SAP System Using a Universal Connector Job Definition File - UNIX
  - Command Line Options
  - Components

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

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

The following figure illustrates the job definition file.

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

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

The following figure illustrates the command to submit the job.

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

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

Command Line Options

Command line options used in this example are:

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

Components

Universal Connector for SAP for UNIX

Running an SAP Job Using SAP Job as Template - UNIX

- Running a Job on an SAP System Using a Pre-existing SAP Job - UNIX
  - Command Line Options
  - Components

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

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

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

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

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

The following figure illustrates the command to run the job.

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

Command Line Options

Command line options used in this example are:

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

Components

Universal Connector for SAP for UNIX

Running an SAP Job Using a Job Definition File - UNIX

- Running a Job on an SAP System Using a Universal Connector Job Definition File - UNIX
  - Command Line Options
  - Components

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

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

Executing this sample will:

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

The following figure illustrates the job definition file.

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

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

The following figure illustrates the command to run the job.

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

Command Line Options

Command line options used in this example are:

Command Options	Description
-dest	Named set of connection parameters (destination) 'CF5'. These connection parameters are used for communications with the SAP system. The "destinations" are stored in file **saprfc.ini**, which must be in the current directory, or its full path must be specified in environment variable **RFC_INI**.
-client	SAP client number that the Universal Connector will communicate with.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-sub	Specification that Universal Connector will issue the SUBMIT command.
-start	Specification that Universal Connector will instruct the SAP system to start the submitted job.
-wait	Specification that Universal Connector will monitor the started job until it completes.
-joblog	Specification that Universal Connector will return the SAP log for the started job.
-spoollist	Specification that Universal Connector will return any spool lists created by the started job.

Components

Universal Connector for SAP for UNIX

Running an SAP Job on a Specific SAP Server - UNIX

- Running a Job on an SAP System on a Specific SAP Server - UNIX
  - Command Line Options
  - Components

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

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

Executing this example will:

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

The following figure illustrates the job definition file.

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

 /* ABAP Step statement. */
```

```
 ABAP_STEP            = "STEP 1"
    ABAP_PROGRAM_NAME = "BTCSPOOL";
```

The following figure illustrates the command to run the job.

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

Command Line Options

Command line options used in this example are:

Command Options	Description
-dest	Named set of connection parameters (destination) 'CF5'. These connection parameters are used for communications with the SAP system. The "destinations" are stored in file **saprfc.ini**, which must be in the current directory, or its full path must be specified in environment variable **RFC_INI**.
-client	SAP client number that the Universal Connector will communicate with.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-sub	Specification that Universal Connector will issue the SUBMIT command.
-start	Specification that Universal Connector will instruct the SAP system to start the submitted job.
-targetserver	Target server for the SAP job to run on.
-wait	Specification that Universal Connector will monitor the started job until it completes.
-joblog	Specification that Universal Connector will return the SAP log for the started job.
-spoollist	Specification that Universal Connector will return any spool lists created by the started job.

Components

Universal Connector for SAP for UNIX

Variant Substitution - UNIX

- Variant Substitution - UNIX
  - Step One
  - Step Two
  - Step Three
  - Command Line Options
  - Components

Variant Substitution - UNIX

This example demonstrates the use of variant substitution.

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

This example is comprised of three steps:

1. Step one modifies SAP variant SBT1.
2. Step two modifies SAP variant SBT2.
3. Step three runs a new SAP job that is created using a pre-existing SAP job as a template.

Variant substitution is performed on the newly created job. As a result, the newly created job will run using the variants that were modified in steps one and two.

Executing this example will:

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

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

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

Step One

This step modifies SAP variant **SBT1**.

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

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

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

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

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

Step Two

This step modifies SAP variant **SBT2**.

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

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

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

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

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

Step Three

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

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

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

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

Command Line Options

Command line options used in this example are:

Command Options	Description
-dest	Named set of connection parameters (destination) 'CF5'. These connection parameters are used for communications with the SAP system. The "destinations" are stored in file **saprfc.ini**, which must be in the current directory, or its full path must be specified in environment variable **RFC_INI**.
-client	SAP client number that the Universal Connector will communicate with.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-sub	Specification that Universal Connector will issue the SUBMIT command.
-start	Specification that Universal Connector will instruct the SAP system to start the submitted job.
-targetserver	Target server for the SAP job to run on.
-wait	Specification that Universal Connector will monitor the started job until it completes.
-joblog	Specification that Universal Connector will return the SAP log for the started job.
-spoollist	Specification that Universal Connector will return any spool lists created by the started job.

Components

Universal Connector for SAP for UNIX

Creating a Variant Definition Using GENERATE VARDEF Command - UNIX

- Creating a USAP Variant Definition Using the USAP GENERATE VARDEF Command - UNIX
  - Command Line Options
  - Components

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

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

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

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

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

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

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

Command Line Options

Command line options used in this example are:

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

Components

Universal Connector for SAP for UNIX

Creating Job Definition Using GENERATE JOBDEF Command - UNIX

- Creating a Universal Connector Job Definition Using the GENERATE JOBDEF Command - UNIX
  - Command Line Options
  - Components

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

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

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

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

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

- Job Name: **USP_TEMPLATE_1**
- Job ID: **12345678**

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

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

Command Line Options

Command line options used in this example are:

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

Components

Universal Connector for SAP for UNIX

Mass Activities Support Example for zOS

- Universal Connector Mass Activity Support Example
  - SYSIN Options for USPVRMS JCL
  - SYSIN Options for USPJRMS JCL
  - Components

Universal Connector Mass Activity Support Example

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

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

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

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

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

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

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

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

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

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

 /* WF\_OKEY */
 SELNAME        = "WF_OKEY"
    KIND        = "P"
```

```
    LOW         = "";

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

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

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

Step 3

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

USPJRMS - JCL

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

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

```
   -pwd    sappwd
   -sub    VARDEF
   -start
   -mawait
/*
```

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

USPPRC - JCL Procedure

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

SYSIN Options for USPVRMS JCL

The SYSIN options used in the USPVRMS JCL example are:

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

SYSIN Options for USPJRMS JCL

The SYSIN options used in the USPJRMS JCL example are:

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

-mawait	Causes USAP to wait for the SAP mass activity jobs to complete processing.

Components

Universal Connector for z/OS

Batch Input Monitoring Example for zOS

Batch Input Processing Example

This example illustrates batch input processing for z/OS.

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

```
//* Run ABAP program RSBDCSUB to perform Batch Input processing
//* using the variant that was modified in step 1.
//*
//* NOTE: This job requires that a variant SBX1 exists for ABAP
//* program RSBDCSUB.
//*
//*****************************************************************
//STEP2    EXEC USPPRC
//JOBDEF   DD  *
 /* Job Header statement. */
 JOBNAME = "RSBDCSUB";
 /* ABAP Step statement. */
 ABAP_STEP            = "STEP 1"
    ABAP_PROGRAM_NAME = "RSBDCSUB_SBX1"
    VARIANT_NAME      = "SBX1";
//SYSIN    DD  *
   -dest    CF5
   -client  800
   -userid  sapuid
   -pwd     sappwd
   -sub     JOBDEF
   -start
   -bdcwait
/*
```

SYSIN Options

SYSIN options used in this example are:

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

Components

Universal Connector for SAP for zOS

Mass Activities Support in Universal Connector

- Mass Activities Support in Universal Connector
  - Mass Activities Process Flow
- Initiating Mass Activities
- Monitoring Mass Activities
- Working with Parameter Records

Mass Activities Support in Universal Connector

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

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

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

Mass Activities Process Flow

The basic process flow in working with mass activities is:

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

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

Initiating Mass Activities

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

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

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

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

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

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

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

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

Monitoring Mass Activities

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

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

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

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

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

Working with Parameter Records

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

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

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

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

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

Batch Input Monitoring in Universal Connector

- Overview
- Batch Input Monitoring Process
- Batch Input Monitoring Requirements
  - SAP System
  - SAP Batch Input Sessions
  - Universal Connector
- Batch Input Monitoring Parameters

Overview

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

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

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

Batch Input Monitoring Process

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

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

Batch Input Monitoring Requirements

SAP System

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

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

SAP Batch Input Sessions

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

Universal Connector

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

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

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

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

Batch Input Monitoring Parameters

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

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

See BDCWAIT for details concerning the use of these parameters.

Universal Data Mover - 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 Universal Data Mover business solution.

With Universal Data Mover, Remote Execution for SAP systems is performed indirectly. Universal Data Mover 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 a Universal Data Mover process.

Universal Data Mover 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 Universal Agent for the Remote Execution for SAP feature of Universal Data Mover. 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

- Raising an SAP Event for z/OS Example
- Raising an SAP Event for UNIX Example

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

Raising an SAP Event for zOS Example

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

The following example demonstrates raising events in a remote SAP system using the Universal Data Mover (UDM) 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.

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

Raising an SAP Event for z/OS JCL

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

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

#*************************************************************\*
```

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

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

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

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

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

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

close
/*
```

Components

Universal Data Mover Manager for z/OS

Universal Data Mover Server for UNIX

Universal Connector for z/OS

Raising an SAP Event for UNIX Example

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

Raising an SAP Event for UNIX Example

The following example demonstrates raising events in a remote SAP system using the Universal Data Mover (UDM) 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.

UDM 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 z/OS

Universal Connector for SAP for zOS

Web Services Execution

- Introduction
- Outbound Implementation
- Inbound Implementation

- Detailed Information

Introduction

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

Outbound Implementation

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

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

Universal Command Agent for SOA gets its payload input from Universal Command through STDIN. When the parameters and data are passed in, the workload execution request is processed and any return data is passed back to Universal Command.

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

Universal Agent enables you to:

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

(See #Examples, below.)

Inbound Implementation

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

This allows for the integration of Internet and message-based applications with systems management functions such as:

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

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

(See Examples, below.)

Detailed Information

The following pages provide detailed information for Web Services Execution:

- Universal Agent - Web Services Examples

Universal Agent - Web Services Examples

- Web Services Outbound Examples - Windows and UNIX
- Web Services Outbound Examples - z/OS
- Web Services Inbound Examples - Windows and UNIX

Web Services Outbound Examples - Windows and UNIX

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

Web Services Outbound Examples - z/OS

- Outbound SOAP Implementation - z/OS

Web Services Inbound Examples - Windows and UNIX

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

Using Universal Agent to Publish to a SOA Workload - Windows and UNIX

- Basic Structure of Using Universal Agent to Publish to a SOA Workload
  - Command Line Options
  - Components
- Example Workloads
  - JMS ActiveMQ Workload
  - JMS Websphere Workload
  - XD Workload
  - MQ Series Workload

Basic Structure of Using Universal Agent to Publish to a SOA Workload

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

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

Command Line Options

The command line options used in this example are:

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

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

Components

Universal Command

Universal Command Agent for SOA

Example Workloads

The following examples illustrate various workloads.

JMS ActiveMQ Workload

```
-jmsdestination dynamicQueues/Soatest2TestQueue1
-jmsconnectionfactoryname ConnectionFactory
-protocol JMS
```

```
-serviceurl tcp://soatest2:61616
-timeoutsec 120
-jmscontextfactoryname org.apache.activemq.jndi.ActiveMQInitialContextFactory
-mep Publish
```

Command Options

The command options used in this example are:

Option	Description
-jmsdestination	Name of the target JMS destination queue or topic for the JMS message.
-jmsconnectionfactoryname	Connection factory to be used to establish a connection to a JMS provider.
-protocol	Message protocol to be used for the current operation.
-serviceurl	URL (internet, network, or file-based) of the target workload.
-timeoutsec	Length of time to wait for the request to complete.
-jmscontextfactoryname	Java class name of the JMS providers initial context factory.
-mep	Message exchange pattern to be used for the current operation.

JMS Websphere Workload

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

Command Options

The command options used in this example are:

Option	Description
-jmsdestination	Name of the target JMS destination queue or topic for the JMS message.
-jmsconnectionfactoryname	Connection factory to be used to establish a connection to a JMS provider.
-protocol	Message protocol to be used for the current operation.
-serviceurl	URL (internet, network, or file-based) of the target workload.
-timeoutsec	Length of time to wait for the request to complete.
-jmscontextfactoryname	Java class name of the JMS providers initial context factory.
-jmspropertiesfile	Name and location of an XML document containing the JMS properties to be included in the JMS message.
-mep	Message exchange pattern to be used for the current operation.

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

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

XD Workload

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

Command Options

The command options used in this example are:

Option	Description
-xdcmd	Operation to submit to the WebSphere XD environment.
-xdcmdid	Used to correlate jobs.
-servicepassword	Password to be passed to the target workload for authentication.
-protocol	Message protocol to be used for the current operation.
-serviceurl	URL (internet, network, or file-based) of the target workload.
-serviceusername	User name to be passed to the target workload for authentication.
-timeoutsec	Length of time to wait for the request to complete.
-mep	Message exchange pattern to be used for the current operation.

MQ Series Workload

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

Command Options

The command options used in this example are:

Option	Description

-mqqueuemanagername	Name of the MQ QUEUE Manager.
-mqqueuename	Name of the MQ Queue to use.
-mqhost	Name of the server running MQSeries.
-mqchannel	Name of the MQ Queue to use.
-protocol	Message protocol to be used for the current operation.
-timeoutsec	Length of time to wait for the request to complete.
-mep	Message exchange pattern to be used for the current operation.

Message Payload for SOAP - Windows and UNIX

- Message Payload for SOAP
  - SOAP Response
  - Components

Message Payload for SOAP

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

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

The first line contains:

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

The second line contains:

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

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

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

SOAP Response

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

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

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

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

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

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

- Logging Configuration
  - Components
  - uac_log4jConfiguration.xml Example
  - uai_log4jConfiguration.xml Example

Logging Configuration

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

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

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

The logging levels supported by the logging implementation are:

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

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

uac_log4jConfiguration.xml Example

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

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

```
    <param name="Source" value="UAC"/>
    <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="%c{1} %M - %m%n"/>
    </layout>
  </appender>
  <appender name="ConsoleAppender" class="org.apache.log4j.ConsoleAppender">
    <param name="Target" value="System.err"/>
    <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="%d %-5p %-17c{2} (%30F:%L) %3x -
    %m%n"/>
    </layout>
  </appender>
  <logger name="com.stonebranch" additivity="true">
    <level value="error"/>
  </logger>
  <root>
    <priority value="error"/>
    <!--<appender-ref ref="LF5Appender"/>-->
    <appender-ref ref="RollingFileAppender"/>
    <!--<appender-ref ref="ConsoleAppender"/>-->
    <!--appender-ref ref="NTEventLogAppender"/--->
  </root>
</log4j:configuration>
```

uai_log4jConfiguration.xml Example

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

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

```
  <root>
    <priority value="error"/>
    <!--appender-ref ref="LF5Appender" / -->
    <!--<appender-ref ref="RollingFileAppender"/>-->
    <appender-ref ref="ConsoleAppender"/>
  </root>
</log4j:configuration>
```

UAC HTTP Form - Windows and UNIX

- Example of Universal Command HTTP POST with Form Data
  - Universal Command Options
  - Components
  - Service Options
  - Form Data

Example of Universal Command HTTP POST with Form Data

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

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

Universal Command Options

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

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

Components

Universal Command

Universal Command Agent for SOA

Service Options

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

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

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

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

Form Data

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

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

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

Outbound SOAP Implementation - zOS

- Outbound SOAP Implementation
  - JCL
  - DD Statements
  - Components

Outbound SOAP Implementation

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

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

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

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

JCL

The following JCL will initiate the outbound SOAP request.

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

This JCL executes the Universal Command JCL procedure.

DD Statements

The DD Statements contain the following:

LOGIN DD

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

SYSIN DD

Universal Command runtime parameters:

- -host
  DNS name or IP address of the Linux Server running Universal Command Agent for SOA.
- -encryptedfile
  Specified the DD name that will contain the encrypted password file.
- -script
  Specifies the DD name that will contain the Universal Command Agent for SOA runtime parameters that are passed to the Universal Command for Agent SOA.
- -script_type
  The value **SERVICE** tells Universal Command that this is a SOA process.

Outbound SOAP Request - SYSIN DD Contents

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

INPUT DD

Universal Command Agent for SOA runtime parameters:

- -protocol
  Indicates which of the supported SOA protocols to use for this request.

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

Outbound SOAP Request - SYSIN DD Contents

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

UNVIN DD

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

Outbound SOAP Request - SYSIN DD Contents

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

Components

Universal Command

Universal Command Agent for SOA

Universal Encrypt

Inbound SOAP Implementation - Windows and UNIX

- Inbound SOAP Implementation
  - Inbound SOAP Request UAC.xml
  - Inbound SOAP Request - Message Payload Written to process_%Seq%.xml File
  - Inbound SOAP Request - Universal Event Monitor Event Definition
  - Loading the Event Definition
  - Changing the Event Definition

- Inbound SOAP Request - Universal Event Monitor Handler Definition
- Outbound SOAP Request - abc.rexx
- Outbound SOAP Request - Event and Handler to purge abc.log
- Components

Inbound SOAP Implementation

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

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

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

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

Inbound SOAP Request UAC.xml

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

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

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

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

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

Inbound SOAP Request - Message Payload Written to process_%Seq%.xml File

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

```
<?xml version='1.0' encoding='utf\-8'?><soapenv:Envelope xmlns:soapenv=
"http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org
/2001/
XMLSchema-instance" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding
/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"><soapenv:Body><NS1:process
xmlns:NS1="http://inbound.uac.stonebranch.com">
<NS1:identitySourceApplicationId>RBS</NS1:identitySourceApplicationId><NS1:
identitySourceUserId />
```

```
<NS1:identitySourcePassword /><NS1:identitySourceToken />
<NS1:activityRequestId>AUT42100211092659702930000</NS1:activityRequestId>
<NS1:activityStatus>PROCESS CLOSE ACCOUNTING YYYY MM</NS1:activityStatus>
<NS1:activityState>ACCOUNTING MONTH CLOSING INPROGRESS</NS1:activityState>
<NS1:activityStateReason>INFO</NS1:activityStateReason>
<NS1:activityAction>ODPT0001</NS1:activityAction>
<NS1:activityStartDate>2010-02-24</NS1:activityStartDate>
<NS1:activityStartTime>08:35:42.397382</NS1:activityStartTime></NS1:process><
/soapenv:Body></soapenv:Envelope>
```

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

- <NS1:identitySourceApplicationId>RBS</NS1:identitySourceApplicationId>
- <NS1:activityRequestId>AUT42100211092659702930000</NS1:activityRequestId>
- <NS1:activityAction>ODPT0001</NS1:activityAction>

Inbound SOAP Request - Universal Event Monitor Event Definition

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

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

  FILESPEC          "/export/home/ control/indesca/soap_listener/*.*"
  MIN_FILE_SIZE
0
  RENAME_FILE
YES
  RENAME_FILESPEC   "/export/home/ control/indesca/soap_listener
/$(origname).$(origext)"
END_EVENT
```

Event Definition Options

The Event Definition options used in this example are:

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

TRIGGERED_ID	ID of an event handler record that UEM will execute when an event occurrence is triggered.
FILESPEC	Name of a file to monitor.
MIN_FILE_SIZE	Size a file must be in order to be considered complete by UEM.
RENAME_FILE	Specifies whether or not UEM should rename a monitored file when an event occurrence is triggered.
RENAME_FILESPEC	Specifies how a file should be renamed when an event occurrence is triggered.

Loading the Event Definition

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

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

Command Line Options

The Event Definition options used in this example are:

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

Changing the Event Definition

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

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

Command Line Options

The Event Definition options used in this example are:

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

Inbound SOAP Request - Universal Event Monitor Handler Definition

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

The following Universal Event Monitor handler definition processes each **Process_%Seq%.xml** file.

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

```
2>&1"
   STMT "set -
xv"

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

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

- uem_handler control,allow,noauth

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

The Event Handler invokes Universal Command to:

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

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

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

Outbound SOAP Request - abc.rexx

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

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

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

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

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

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

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

EXIT ABC.RC
```

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

- ucmd_access ALL,*,CTLMNT,allow,noauth

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

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

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

Outbound SOAP Request - Event and Handler to purge abc.log

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

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

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

Event Options

The Event options used in this example are:

Option	Description
EVENT_ID	Identifier that uniquely identifies an event definition record.

EVENT_TYPE	Type of system event represented by the event definition record.
COMP_NAME	Event-driven UEM Server responsible for monitoring the event.
STATE	Event definitions that should be processed or ignored by UEM.
TRACKING_INT	Event definitions that should be processed or ignored by UEM.
TRIGGERED_ID	ID of an event handler record that UEM will execute when an event occurrence is triggered.
FILESPEC	Name of a file to monitor.
MIN_FILE_SIZE	Size a file must be in order to be considered complete by UEM.
HANDLER_ID	Identifier that uniquely identifies an event handler record.
HANDLER_TYPE	Type of process executed for the event handler.
MAXRC	Highest value with which a handler can exit to still be considered as having executed successfully.
USERID	ID of a user account in whose security context the handler process will be executed.
PWD	Password for the user account specified by userid.
CMD	Command to execute on behalf of the event handler.

Components

Universal Event Monitor

UEMLoad

Universal Event Monitor for SOA

Universal Broker

Universal Write-to-Operator

Inbound JMS Implementation - Windows and UNIX

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

Inbound JMS Implementation

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

The following figure illustrates an example of this construction.

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

In the following examples:

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

ActiveMQ Topic

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

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

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

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

Websphere Queue

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

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

<sb:Value>com.ibm.websphere.naming.WsnInitialContextFactory</sb:Value>
            </sb:Property>
            <sb:Property>
                <sb:Name>java.naming.provider.url</sb:Name>
                <sb:Value>iiop://soatest2:2809</sb:Value>
                   </sb:Property>
            <sb:Property>
                <sb:Name>com.ibm.CORBA.ORBInit</sb:Name>
                <sb:Value>com.ibm.ws.sib.client.ORB</sb:Value>
            </sb:Property>
        </sb:InitialContextProperties>
        <sb:ConnectionFactory>jms/SBSConnectionFactory</sb:ConnectionFactory>
        <sb:Listeners>
            <sb:JMSListener>
```

```
                <sb:Destination>jms/Soatest2TestQueue3</sb:Destination>
                <sb:Actions>
                    <sb:JMSFileWriter>
                        <sb:Directory>filesystem<sb:Directory>
            <sb:FilenamePattern>Websphere_Queue_%Seq%.txt</sb:
FilenamePattern>
                        <sb:StartSequenceNumber>1</sb:StartSequenceNumber>
                        <sb:WriteProperties>false</sb:WriteProperties>
                    </sb:JMSFileWriter>
              </sb:Actions>
         </sb:JMSListener>
    </sb:Listeners>
</sb:JMSConnection>
```

MQ Series Queue

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

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

Triggering an Event

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

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

```
begin_event
 event_id "JMS_MESSAGE_TRIGGER"
 event_type FILE
 comp_name uems
 state enable
 tracking_int 10
 triggered_id "JMS_MESSAGE_HANDLER"
 filespec "filesystem/*.txt"
 min_file_size 0
 rename_file yes
```

```
 rename_filespec "filesystem/$(origname).xml"
end_event

begin_handler
 handler_id "JMS_MESSAGE_HANDLER"
 handler_type CMD
 maxrc 0
 userid username
 pwd user_password
 cmd "move $(origname).xml $(origname).found"
end_handler
```

Event Options

The Event options used in this example are:

Option	Description
event_id	Identifier that uniquely identifies an event definition record.
event_type	Type of system event represented by the event definition record.
comp_name	Event-driven UEM Server responsible for monitoring the event.
state	Event definitions that should be processed or ignored by UEM.
tracking_int	Event definitions that should be processed or ignored by UEM.
triggered_id	ID of an event handler record that UEM will execute when an event occurrence is triggered.
filespec	Name of a file to monitor.
min_file_size	Size a file must be in order to be considered complete by UEM.
rename_file	Specifies whether or not UEM should rename a monitored file when an event occurrence is triggered.
rename_filespec	Specifies how a file should be renamed when an event occurrence is triggered.
handler_id	Identifier that uniquely identifies an event handler record.
handler_type	Type of process executed for the event handler.
maxrc	Highest value with which a handler can exit to still be considered as having executed successfully.
userid	ID of a user account in whose security context the handler process will be executed.
pwd	Password for the user account specified by userid.
cmd	Command to execute on behalf of the event handler.

Components

Universal Event Monitor

UEMLoad

Universal Event Monitor for SOA

Universal Data Mover - Web Services Execution

Web Services Execution (Inbound Implementation)

The inbound implementation of Universal Agent's web services execution – Universal Event Monitor for SOA – provides Universal Data Mover 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
  - ActiveMQ Topic
  - Websphere Queue
  - MQ Series Queue
  - Triggering an Event
  - Components

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 Request UAC.xml (UNIX)
  - Inbound SOAP Request UAC.xml (Windows)
  - Inbound SOAP Request - Message Payload Written to process_%Seq%.xml File
  - Inbound SOAP Request - Universal Event Monitor Event Definition
  - Loading the Event Definition
  - Changing the Event Definition
  - Inbound SOAP Request - Universal Event Monitor Handler Definition
  - Outbound SOAP Request - abc.rexx
  - Outbound SOAP Request - Event and Handler to purge abc.log
  - Components

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 (UNIX)

```
<?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">
  <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>false</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 UAC.xml (Windows)

```
<?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">
  <sb:SOAPConnection>
        <sb:URI>/axis2/services/UACInbound</sb:URI>
        <sb:Listeners>
            <sb:SOAPListener>
                <sb:Operation>process</sb:Operation>
                <sb:Actions>
                    <sb:SOAPFileWriter>
                        <sb:Directory>c:\tmp\</sb:Directory>
                        <sb:FilenamePattern>process_%Seq%.xml</sb:
FilenamePattern>
                        <sb:StartSequenceNumber>1</sb:StartSequenceNumber>
                        <sb:WriteEnvelope>false</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:

`c:\tmp\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.org2001
/XMLSchema-instance" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding
/" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soapenv:Body>
<NS1:process xmlns:NS1="http://inbound.uac.stonebranch.com">
<NS1:identitySourceApplicationId>RBS</NS1:identitySourceApplicationId><NS1:
identitySourceUserId />
<NS1:identitySourcePassword /><NS1:identitySourceToken />
<NS1:activityRequestId>AUT4210021109265970293000</NS1:activityRequestId>
<NS1:activityStatus>PROCESS CLOSE ACCOUNTING YYYY MM</NS1:activityStatus>
<NS1:activityState>ACCOUNTING MONTH CLOSING INPROGRESS</NS1:activityState>
<NS1:activityStateReason>INFO</NS1:activityStateReason>
<NS1:activityAction>ODPT0001</NS1:activityAction>
<NS1:activityStartDate>2010-02-24</NS1:activityStartDate>
<NS1:activityStartTime>08:35:42.397382</NS1:activityStartTime>
</NS1:process>
</soapenv:Body>
</soapenv:Envelope>
```

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

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

Inbound SOAP Request - Universal Event Monitor Event Definition

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

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

```
10
  TRIGGERED_ID      "ABC SOA
HANDLER"

  FILESPEC          "/export/home/ control/indesca/soap_listener/*.*"
  MIN_FILE_SIZE
0
  RENAME_FILE
YES
  RENAME_FILESPEC   "/export/home/ control/indesca/soap_listener
/$(origname).$(origext)"
END_EVENT
```

Event Definition Options

The Event Definition options used in this example are:

Option	Description
EVENT_ID	Identifier that uniquely identifies an event definition record.
EVENT_TYPE	Type of system event represented by the event definition record.
COMP_NAME	Event-driven UEM Server responsible for monitoring the event.
STATE	Event definitions that should be processed or ignored by UEM.
TRACKING_INT	Event definitions that should be processed or ignored by UEM.
TRIGGERED_ID	ID of an event handler record that UEM will execute when an event occurrence is triggered.
FILESPEC	Name of a file to monitor.
MIN_FILE_SIZE	Size a file must be in order to be considered complete by UEM.
RENAME_FILE	Specifies whether or not UEM should rename a monitored file when an event occurrence is triggered.
RENAME_FILESPEC	Specifies how a file should be renamed when an event occurrence is triggered.

Loading the Event Definition

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

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

Command Line Options

The Event Definition options used in this example are:

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

Changing the Event Definition

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

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

Command Line Options

The Event Definition options used in this example are:

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

Inbound SOAP Request - Universal Event Monitor Handler Definition

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

The following Universal Event Monitor handler definition processes each **Process_%Seq%.xml** file.

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

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

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

- uem_handler control,allow,noauth

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

The Event Handler invokes Universal Command to:

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

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

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

Outbound SOAP Request - abc.rexx

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

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

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

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

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

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

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

EXIT ABC.RC
```

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

- ucmd_access ALL,*,CTLMNT,allow,noauth

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

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

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

Outbound SOAP Request - Event and Handler to purge abc.log

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

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

```
  MIN_FILE_SIZE
10M
END_EVENT

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

Event Options

The Event options used in this example are:

Option	Description
EVENT_ID	Identifier that uniquely identifies an event definition record.
EVENT_TYPE	Type of system event represented by the event definition record.
COMP_NAME	Event-driven UEM Server responsible for monitoring the event.
STATE	Event definitions that should be processed or ignored by UEM.
TRACKING_INT	Event definitions that should be processed or ignored by UEM.
TRIGGERED_ID	ID of an event handler record that UEM will execute when an event occurrence is triggered.
FILESPEC	Name of a file to monitor.
MIN_FILE_SIZE	Size a file must be in order to be considered complete by UEM.
HANDLER_ID	Identifier that uniquely identifies an event handler record.
HANDLER_TYPE	Type of process executed for the event handler.
MAXRC	Highest value with which a handler can exit to still be considered as having executed successfully.
USERID	ID of a user account in whose security context the handler process will be executed.
PWD	Password for the user account specified by userid.
CMD	Command to execute on behalf of the event handler.

Components

Universal Event Monitor

UEMLoad

Universal Event Monitor for SOA

Universal Broker

Universal Write-to-Operator

Copying Files to and from Remote Systems

- Introduction

- Copying Files Examples - z/OS
- Copying Files Examples - Windows
- Copying Files Examples - UNIX
- Copying Files Examples - IBM i

Introduction

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

The following examples illustrate file copying for all supported platforms.

Copying Files Examples - z/OS

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

Copying Files Examples - Windows

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

Copying Files Examples - UNIX

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

Copying Files Examples - IBM i

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

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

Copy from Local zOS to Remote Windows

- Copy from Local z/OS to Remote Windows via Universal Copy
  - SYSIN Options
  - Components

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

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

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

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

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

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

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

Universal Command Manager for z/OS

Universal Copy

Copy from Remote Windows to Local zOS

- Copy from Remote Windows to Local z/OS via Universal Copy
  - SYSIN Options
  - Components

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

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

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

```
-encryptedfile LOGONDD
-host dallas
/*
```

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

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

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

Universal Command Manager for z/OS

Universal Copy

Copy from Local zOS to Remote UNIX

- Copy from Local z/OS to Remote UNIX via Universal Copy
  - SYSIN Options
  - Components

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

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

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

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

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

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

Universal Command Manager for z/OS

Universal Copy

Copy from Remote UNIX to Local zOS

- Copy from Remote UNIX to Local z/OS via Universal Copy
  - SYSIN Options
  - Components

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

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

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

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

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

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

Universal Command Manager for z/OS

Universal Copy

Copy from Local zOS to Remote IBM i

- Copy from Local z/OS to Remote IBM i via Universal Copy
  - SYSIN Options
  - Components

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

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

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

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

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

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

Universal Command Manager for z/OS

Universal Copy

Copy from Remote IBM i to Local zOS

- Copy from Remote IBM i to Local z/OS via Universal Copy
  - SYSIN Options
  - Components

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

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

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

```
//SYSIN    DD *
-script SCRIPTDD
-encryptedfile LOGONDD
-host dallas
/*
```

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

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

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

Universal Command Manager for z/OS

Universal Copy

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

- Third-Party Copy via Local z/OS, from Windows to UNIX via Universal Copy
  - Parameters
  - SYSIN Options
  - Components

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

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

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

```
SET RC=%ERRORLEVEL%
del %TEMPUNIXID%
URC %RC%
//SYSIN    DD *
-script SCRIPT
-encryptedfile
LOGONDD
-host NTHOST
-level info
/*
```

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

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

Parameters

The following parameters should be changed to match your information:

Parameter	Description
#USERIDUNX	Encrypted userid and password member for UNIX server
#USERIDNT	Encrypted userid and password member for NT server
UCOPYPATH	Path to UCOPY on the receiving UNIX server
OUTPUTFILE	Path and filename of receiving file on UNIX server
INPUTFILE	Path and file name of sending file on Windows server
UNIXHOST	IP address or hostname of receiving UNIX server
NTHOST	IP address or hostname of sending Windows server
TEMPUNIXID	Temporary file on the Windows server used to house the encrypted logon information for the UNIX server. This file is deleted at the bottom of the script.
MODE	Mode of file transfer (binary/text).

SYSIN Options

The SYSIN options used in this example are:

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

Components

Universal Command Manager for z/OS

Universal Command Manager for Windows

Universal Command Server for UNIX

Universal Copy

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

- Third-Party Copy via Local z/OS, from UNIX to Windows via Universal Copy
  - Parameters
  - SYSIN Options
  - Components

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

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

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

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

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

Parameters

The following parameters should be changed to match your information:

Parameter	Description
#USERIDUNX	Encrypted userid and password member for UNIX server
#USERIDNT	Encrypted userid and password member for Windows server
UCOPYPATH	Path to UNIX **ucopy** executable
UCMDPATH	Path to UNIX **ucmd** executable
OUTPUTFILE	Path and filename of receiving file
INPUTFILE	Path and file name of sending file
NTHOST	IP address or hostname of receiving Windows server
TEMPNTID	Temporary file on the UNIX server used to house the encrypted logon information for the Windows server.

MODE	Mode of file transfer (binary / text). Default is set to text.

SYSIN Options

The SYSIN options used in this example are:

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

Components

Universal Command Manager for z/OS

Universal Command Manager for UNIX

Universal Command Server for Windows

Universal Copy

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

- Third-Party Copy via Local z/OS, from Windows to Windows via Universal Copy
  - Parameters
  - SYSIN Options
  - Components

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

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

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

```
//S1 EXEC UCMDPRC
//UNVIN    DD DSN=hlq.userid(#nt2logon),DISP=SHR
//LOGONDD  DD DSN=hlq.userid(#ntlogon),DISP=SHR
//UNVERR   DD DSN=hlq.output(stderr),DISP=SHR
//SCRIPT   DD *
@ECHO ON
:: TRANSFER FROM NT to NT
@SET OUTPUTFILE=c:\temp\output.file
@SET INPUTFILE=c:\temp\input.file
@SET NT2HOST=hostname
@SET TEMPNT2ID=c:\temp\userid.enc
@SET MODE=text
ucopy -output %TEMPNT2ID%
ucmd¬
 -cmd "ucopy -output %OUTPUTFILE%" < %INPUTFILE% ¬
 -host %NT2HOST% -encryptedfile %TEMPNT2ID% -level info -stdin -mode %MODE%
SET RC=%ERRORLEVEL%
del %TEMPNT2ID%
URC %RC%
//SYSIN    DD  *
 -script SCRIPT
```

```
 -encryptedfile LOGONDD
 -host NTHOST
 -level info   /*
//*
//*************************************************************
//S1 EXEC PGM=UMET,PARM='-TABLE TABLE -LEVEL VERBOSE'
//STEPLIB  DD  DISP=SHR,DSN=hlq.UNV.SUNVLOAD
//SYSPRINT DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
//TABLE    DD  *
  "The system cannot find the file specified." 8
/*
//SYSIN    DD  DISP=SHR,DSN=hlq.output(stderr)
//*************************************************************
//S1 EXEC PGM=IEBGENER
//SYSUT1   DD DISP=SHR,DSN=hlq.output(stderr)
//SYSUT2   DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSIN    DD DUMMY
```

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

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

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

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

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

Parameters

The following parameters should be changed to match your information:

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

Universal Command Manager for z/OS

Universal Command Manager for Windows

Universal Command Server for Windows

Universal Copy

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

- Third-Party Copy via Local z/OS, from UNIX to UNIX via Universal Copy
  - Parameters
  - SYSIN Options
  - Components

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

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

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

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

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

Parameters

The following parameters should be changed to match your information:

Parameter	Description
UCOPYPATH	Path pointing to the **ucopy** executable on the second UNIX server
UCMDPATH	Path pointing to the **ucmd** executable on the second UNIX server
OUTPUTFILE	Path and filename of receiving file
INPUTFILE	Path and file name of sending file
UNIXSHOST	IP address or hostname of sending UNIX server
UNIXRHOST	IP address or hostname of receiving UNIX server
TEMPUNIXRID	Temporary file on the sending UNIX server used to house the encrypted logon information for the receiving UNIX server.
MODE	Mode of file transfer (binary / text). Default is set to text.
USERUNXR	Points to the userid / password information for the receiving UNIX server.
USERUNXS	Points to the userid / password information for the sending UNIX server.

SYSIN Options

The SYSIN options used in this example are:

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

Components

Universal Command Manager for z/OS

Universal Command Manager for UNIX

Universal Command Server for UNIX

Universal Copy

Copy from Local zOS to Remote System (in Binary)

- Copy from Local z/OS to Remote System (in Binary) via Universal Copy
  - SYSIN Options
  - Components

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

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

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

```
-stdin -mode binary
/*
```

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

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

Universal Command Manager for z/OS

Universal Copy

Copy from Remote System to Local zOS (in Binary)

- Copy from Remote System to Local z/OS (in Binary) via Universal Copy
  - SYSIN Options
  - Components

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

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

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

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

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

Universal Command Manager for z/OS

Universal Copy

Copy from Local zOS to Remote zOS

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

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

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

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

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

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

The file is copied as a text file (default transfer mode for standard files is text).

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	DD from which to read a script file. The script file is sent to the remote system for execution.
-encryptedfile	DD from which to read an encrypted command options file.
-host	Directs the command to a computer with a host name of **dallas**.
-stdin	Specification that the options following this one apply to the stdin file.
-encrypt	Specification that standard file data sent over the network is encrypted.
-compress	Specification for whether the standard file data transmitted across the network should be compressed.

-authenticate	Specification that standard file data sent over the network is authenticated.

Components

Universal Command Manager for z/OS

Universal Command Server for z/OS

Universal Copy

Copy from Remote zOS to Local zOS

- Copy from Remote z/OS to Local z/OS (with Encryption, Compression, and Data Authentication) via Universal Copy
  - SYSIN Options
  - Components

Copy from Remote z/OS to Local z/OS (with Encryption, Compression, and Data Authentication) via Universal Copy

The following figure illustrates the copying of a file from a remote z/OS system to a local z/OS system (with encryption, compression, and data authentication).

```
//STEP1 EXEC UCMDPRC
//UNVOUT='DISP=SHR,DSN=MY.PDS(MEMBER)
//LOGONDD  DD DISP=SHR,DSN=MY.LOGON(USERID)
//SCRIPTDD DD *
/opt/universal/bin/ucopy < //'REMOTE.PDS(MEMBER)'
//SYSIN DD *
-script SCRIPTDD
-encryptedfile LOGONDD
-host dallas
-stdout -encrypt yes -compress yes -authenticate yes
/*
```

The JCL procedure **UCMDPRC** is used to execute the command. The command is sent to a remote system named **dallas** for execution. The **UN VOUT** DD specifies a local data set to use for the standard output of the remote command.

Options are read from the encrypted file allocated to DD **LOGONDD**.

The file is copied as a text file, since the default transfer mode for standard files is text.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	DD from which to read a script file. The script file is sent to the remote system for execution.
-encryptedfile	DD from which to read an encrypted command options file.
-host	Directs the command to a computer with a host name of **dallas**.
-stdout	Specification that the options following this one apply to the stdout file.
-encrypt	Specification that standard file data sent over the network is encrypted.
-compress	Specification for whether the standard file data transmitted across the network should be compressed.
-authenticate	Specification that standard file data sent over the network is authenticated.

Components

Universal Command Manager for z/OS

Universal Command Server for z/OS

Universal Copy

Copy from Local zOS to Remote Windows (with Windows Date Variables)

- Copy from Local z/OS to Remote Windows (with Windows Date Variables) via Universal Copy
  - SYSIN Options
  - Components

Copy from Local z/OS to Remote Windows (with Windows Date Variables) via Universal Copy

The following figure illustrates the copying of a file from a local z/OS system to a remote Windows system.

The file name on the Windows server is dynamically created based on the current date.

```
//S1 EXEC UCMDPRC
//UNVIN    DD DISP=SHR,DSN=hlq.input.file
//LOGONDD  DD DISP=SHR,DSN=hlq.userid(userid)
//SCRIPTDD DD *
@echo off
for /f "tokens=1 delims=/" %%a in ('date /t') do set daymm=%%a
for /f "tokens=2" %%a in ('echo %daymm%') do set mm=%%a
for /f "tokens=2 delims=/" %%a in ('date /t') do set dd=%%a
for /f "tokens=3 delims=/" %%a in ('date /t') do set yy=%%a
echo daymm:  %daymm%
echo mmddyy: %mm%%dd%%yy%
ucopy -output c:\temp\outputfile%mm%%dd%%yy%
//SYSIN    DD *
-script SCRIPTDD
-encryptedfile LOGONDD
-host dallas
/*
```

The JCL procedure **UCMDPRC** is used to execute the command. The command is sent to a remote system named **dallas** for execution. The -output option is used with the **ucopy** command to direct stdout to a local data set on the remote Windows server. The file name is created with a date variable. The date variable is set to the current date in the commands preceding the **ucopy** command.

Additional command line options are read from the encrypted file allocated to DD **LOGONDD**.

The file is copied as a text file, since the default transfer mode for standard files is text.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	DD from which to read a script file. The script file is sent to the remote system for execution.
-encryptedfile	DD from which to read an encrypted command options file.
-host	Directs the command to a computer with a host name of **dallas**.

Components

Universal Command Manager for z/OS

Universal Command Server for Windows

Universal Copy

Copy from Local zOS to Remote UNIX (with UNIX Date Variables)

- Copy from Local z/OS to Remote UNIX (with UNIX Data Variables) via Universal Copy
  - SYSIN Options

- Components

Copy from Local z/OS to Remote UNIX (with UNIX Data Variables) via Universal Copy

The following figure illustrates the copying of a file from a local z/OS system to a remote UNIX system. The file name on the UNIX server is dynamically created based on the current date.

```
//S1 EXEC UCMDPRC
//UNVIN    DD DISP=SHR,DSN=hlq.input.file
//LOGONDD  DD DISP=SHR,DSN=hlq.userid(userid)
//SCRIPTDD DD *
DATEN=`date +%d%m`
export DATEN
echo $DATEN
/opt/universal/bin/ucopy \
-output /tmp/output$DATEN.file
```

The JCL procedure **UCMDPRC** is used to execute the command. The command is sent to a remote system named **dallas** for execution. The stdout redirection character **>** is used with the **ucopy** command to direct stdout to a local data set on the remote server. The file name is created with a date variable, which is set to the current date in the commands preceding the **ucopy** command. The path to the **ucopy** binary must be specified if the directory is not defined in the user's path environmental variable.

Additional command line options are read from the encrypted file allocated to DD **LOGONDD**. The UNIX continuation character **** is used to split the **ucopy** command to two lines.

The file is copied as a text file, since the default transfer mode for standard files is text.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	DD from which to read a script file. The script file is sent to the remote system for execution.
-encryptedfile	DD from which to read an encrypted command options file.
-host	Directs the command to a computer with a host name of **dallas**.

Components

Universal Command Manager for z/OS

Universal Command Server for UNIX

Universal Copy

Copy from Remote UNIX to Local zOS Using cat Command

- Copy from Remote UNIX to Local z/OS Using UNIX cat Command via Universal Command Manager for z/OS
  - SYSIN Options
  - Components

Copy from Remote UNIX to Local z/OS Using UNIX cat Command via Universal Command Manager for z/OS

The following figure illustrates the copying of a file from a remote UNIX system to a local z/OS system using the UNIX **cat** command.

```
//UNIXCAT  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*
//         JCLLIB ORDER=username.UNV.SUNVSAMP
//*
//STEP1    EXEC UCMDPRC,
//              STDOUT='DISP=SHR,DSN=username.UNIX.FILE1'
```

```
//SYSIN    DD  *
-cmd 'cat /export/home/username/file1'
-host Unix_1 -userid username -pwd password -level audit
/*
```

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-cmd	Remote command **`cat /export/home/username/file1`** to execute. The **cat** program copies the files specified on the command line to its **stdout.n**.
-host	Directs the command to a computer with host address **Unix_1**.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-level	Message level output for this command execution.

Components

Universal Command Manager for z/OS

Copy from Remote UNIX to Local Windows

- Copy from Remote UNIX to Local Windows via Universal Command Manager for Windows
  - Parameters
  - Command Line Options
  - Components

Copy from Remote UNIX to Local Windows via Universal Command Manager for Windows

The following figure illustrates the copying of a file from a remote UNIX system to a local Windows system. Although the command is shown on two lines, it should be entered on one line at the command prompt. If it is coded in a script, the Windows continuation character of ¬ must be used.

```
ucmd -cmd " /opt/universal/bin/ucopy unixinputfile"
-host unixhost -encryptedfile unixid.file > c:\temp\ntoutputfile
```

The standard out of the **ucopy** command on the remote host is redirected back to the local host and written to **`c:\temp\ntoutput.file`**. The command **ucopy** is installed as part of Universal Command Server on the remote system.

The file is copied as a text file, since the default transfer mode for standard files is text.

Parameters

The following parameters should be changed to match your information:

Parameter	Description
ntoutputfile	Path and filename of output file
unixinputfile	Path and file name of input file
unixhost	IP address of remote UNIX server
unixid.file	File on the Windows server used to house the authentication parameters for the UNIX server

Command Line Options

The command line options used in this example are:

Option	Description
-cmd	Remote command **ucopy file** to execute. The **ucopy** program copies the files specified on the command line to its STDOUT.
-host	Directs the command to a computer with a host name of **unixhost**.
-encryptedfile	File from which to read encrypted command options.

Components

Universal Command Manager for Windows

Universal Copy

Copy From Local Windows to Remote UNIX

- Copy From Local Windows to Remote UNIX via Universal Command Manager for Windows
  - Parameters
  - Command Line Options
  - Components

Copy From Local Windows to Remote UNIX via Universal Command Manager for Windows

The following figure illustrates the copying of a file from a local Windows system to a remote UNIX system. Although the command is shown on two lines, it should be entered on one line at the command prompt. If it is coded in a script, the Windows continuation character of ¬ must be used.

```
ucmd -cmd " /opt/universal/bin/ucopy -output /tmp/unixoutputfile"
-host unixhost -encryptedfile unixid.file
< c:\temp\ntinputfile
```

The stdin of the **ucmd** manager on the local host is redirected to the stdout of the remote host and written to **`/tmp/unixoutputfile`**. The command **ucopy** is installed as part of Universal Command Server on the remote system. The file is copied as a text file since the default transfer mode for standard files is text.

Parameters

The following parameters should be changed to match your information:

Parameter	Description
unixoutputfile	Path and filename of output file
ntinputfile	Path and file name of input file
unixhost	IP address of sending UNIX server
unixid.file	File on the Windows server used to house the authentication parameters for the UNIX server

Command Line Options

The command line options used in this example are:

Option	Description
-cmd	Remote command **ucopy file** to execute. The **ucopy** program copies the files specified on the command line to its stdout.
-host	Directs the command to a computer with a host name of **unixhost**.
-encryptedfile	File from which to read encrypted command options.

Components

Universal Command Manager for Windows

Universal Copy

Copy from Remote UNIX to Local Windows Using the UNIX cat Command

- Copy from Remote UNIX to Local Windows Using the UNIX cat Command via Universal Command Manager for Windows
  - Command Line Options
  - Components

Copy from Remote UNIX to Local Windows Using the UNIX cat Command via Universal Command Manager for Windows

The following figure illustrates copying of file from a remote UNIX system to a local Windows system using the UNIX **cat** command.

Although the command is shown on two lines, it should be entered as one line at the command prompt.

```
ucmd -cmd "cat ~/file" -host dallas
     -userid joe -pwd password -comment "copy ~/file from dallas" > localfile
```

The stdout of the **cat** command on the remote host is redirected back to the local host and written to the stdout of **ucmd**, which is then redirected to the local file **localfile**.

Command Line Options

The command line options used in this example are:

Option	Description
-cmd	Remote command - **"cat ~/file"** - to execute. The **cat** program copies the files specified on the command line to its stdout.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-comment	Description of the process executed by Universal Command.

The file is copied as a text file, since the default transfer mode is **text**.

Components

Universal Command Manager for UNIX

Universal Copy

Copy from Local UNIX to Remote Windows

- Copy from Local UNIX to Remote Windows via Universal Command Manager for UNIX
  - Command Line Options
  - Components

Copy from Local UNIX to Remote Windows via Universal Command Manager for UNIX

The following figure illustrates the copying of a file from a local UNIX system to a remote Windows system.

Although the command is shown on two lines, it should be entered as one line at the command prompt.

```
ucmd -cmd 'ucopy > remotefile' -host dallas
     -userid joe -pwd password < localfile
```

The **ucopy** command receives its stdin file from ucmd. The standard in of UCMD is redirected from **localfile**.

The command **ucopy** is installed as part of Universal Command Server on the remote system.

The file is copied as a text file, since the default transfer mode is **text**.

Command Line Options

The command line options used in this example are:

Option	Description
-cmd	Remote command **`ucopy > remotefile`** to execute. The **ucopy** program copies its standard in to its standard out.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

Components

Universal Command Manager for UNIX

Universal Copy

Copy Encrypted File from Local UNIX to Remote Windows

- Copy Encrypted File from Local UNIX to Remote Windows via Universal Command Manager for UNIX
  - Command Line Options
  - Components

Copy Encrypted File from Local UNIX to Remote Windows via Universal Command Manager for UNIX

The following figure illustrates the copying of a file from a local UNIX system to a remote Windows server.

Although the command is shown on two lines, it should be entered on one line at the command prompt. If it is coded in a script, the UNIX continuation character of **** must be used.

```
ucmd -cmd 'ucopy -output c:\temp\ntoutput.file' -host nthost
     -encryptedfile login.file < /tmp/unixinput.file
```

The stdin of the ucmd manager on the local host is redirected to the remote host and written to stdout file **`c:\temp\ntoutput.file`**. The command **ucopy** is installed as part of Universal Command Server on the remote system.

The file is copied as a text file, since the default transfer mode for standard files is text.

Command Line Options

The command line options used are:

Option	Description
-cmd	Remote command **`ucopy file`** to execute. The **ucopy** program copies the files specified on the command line to its stdout.
-host	Directs the command to a computer with a host name of **nthost**.
-encryptedfile	File from which to read encrypted command options.

Components

Universal Command Manager for UNIX

Universal Copy

Copy from Remote Windows to Local UNIX

- Copy from Remote Windows to Local UNIX via Universal Command Manager for UNIX
  - Command Line Options

- Components

Copy from Remote Windows to Local UNIX via Universal Command Manager for UNIX

The following figure illustrates the copying of a file from a remote Windows system to a local UNIX system.

Although the command is shown on two lines, it should be entered as one line at the command prompt.

```
ucmd -cmd 'ucopy file' -host dallas
     -userid joe -pwd password > localfile
```

The stdout of the **ucopy** command on the remote host is redirected back to the local host and written to the stdout of ucmd, which is then redirected to the local file **localfile**.

The command **ucopy** is installed as part of UCMD Server on the remote system.

The file is copied as a text file since the default transfer mode is **text**.

Command Line Options

The command line options used in this example are:

Option	Description
-cmd	Remote command **`ucopy file`** to execute. The **ucopy** program copies the files specified on the command line to its stdout.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

Components

Universal Command Manager for Windows

Universal Copy

Copy Encrypted File from Remote Windows to Local UNIX

- Copy Encrypted File from Remote Windows to Local UNIX via Universal Command Manager for UNIX
  - Command Line Options
  - Components

Copy Encrypted File from Remote Windows to Local UNIX via Universal Command Manager for UNIX

The following figure illustrates the copying of a file from a remote Windows system to a local UNIX server.

If it is coded in a script, then the UNIX continuation character of **** must be used.

```
ucmd -cmd 'ucopy ntinputfile' -host nthost -encryptedfile
ntid.file > /tmp/unixoutputfile
```

The stdout of the **ucopy** command on the remote host is redirected back to the local host and written to **`/tmp/unixoutputfile`**. The command **ucopy** is installed as part of Universal Command Server on the remote system.

The file is copied as a text file, since the default transfer mode for standard files is text.

Command Line Options

The command line options used are:

Option	Description

-cmd	Remote command **ucopy file** to execute. The **ucopy** program copies the files specified on the command line to its stdout.
-host	Directs the command to a computer with a host name of **nthost**.
-encryptedfile	File from which to read encrypted command options.

Components

Universal Command Manager for Windows

Universal Copy

Copy from Remote Windows to Local IBM i via UCMD Manager

- Copy from Remote Windows to Local IBM i via Universal Command Manager for IBM i
  - Command Line Options
  - Components

Copy from Remote Windows to Local IBM i via Universal Command Manager for IBM i

The following figures illustrate copying a file from a remote Windows system to a local file.

```
STRUCM CMD('ucopy infile') HOST(dallas) USERID(joe) PWD(password) SOTFILE
(localfile)
```

UCOPY, which the UCMD Server invokes on system **dallas**, retrieves data from the file named **infile**. It sends data to the UCMD Manager running under IBM i using standard output. The UCMD Manager, in turn, receives input via stdin and writes to file specified by SOTFILE, **localfile**. The file is copied via UCMD as a text file, since the default transfer mode is **text**.

```
STRUCM CMD('ucopy c:\ntinput.file') HOST(nthost) USERID(joe) PWD(akkSdiq)
SOTFILE(library/outputfile) SOTMBR(member)
```

UCOPY runs on the remote host and retrieves data from **c:\ntinput.file**. UCOPY output is redirected back to the local host and written to the stdout of **STRUCM**. **STRUCM** output is, in turn, directed to the local file **SOTFILE** and, optionally, **SOTMBR**. The file is copied via UCMD as a text file, since the default transfer mode for standard files is text.

The command **ucopy** is installed as part of UCMD Server on the remote system.

Command Line Options

The command line options used are:

Option	Description
CMD	Remote command to execute. The **ucopy** program copies the files specified on the command line to its stdout.
HOST	Directs the command to a computer with a host name of **nthost**.
USERID	Remote user ID with which to execute the command.
PWD	Password for the user ID.
SOTFILE [SOTMBR]	Location to which the stdout file data is written.

Components

Universal Command Manager for IBM i

Universal Command Server for Windows

Universal Copy

Copy from Remote IBM i to Local Windows via UCMD Manager

- Copy from Remote IBM i to Local Windows via Universal Command Manager for IBM i
  - Command Line Options
  - Components

Copy from Remote IBM i to Local Windows via Universal Command Manager for IBM i

The following figure illustrates the copying of a file initiated by Windows, which copies the first member of a file from IBM i to a file on the Windows system.

```
ucmd -host sysName -userid userId -pwd password
-cmd "strucp frmfile(mylib/myfile)" > D:\tmp\File400.txt
```

UCMD running on Windows invokes STRUCP via a UCMD Server running on IBM i. The FRMFILE parameter overrides input from stdin to the file **`mylib/file`**. Since the FRMMBR parameter is not used, input defaults to the file member ***FIRST**. Data is transferred from **`mylib/myfile`** `to` **`D:\tmp\File400.txt`** via UCMD Manager stdout.

The command **STRUCP** is installed as part of UCMD Server on the IBM i system.

The file is copied as a text file, since the default transfer mode is **text**.

Command Line Options

The command line options used in this example are:

Option	Description
-cmd	Remote command to execute on the IBM i.
-host	Directs the command to a computer with a host name of **sysName**.
-userid	IBM i user ID with which to execute the command.
-pwd	Password for the user ID.

Components

Universal Command Manager for Windows

Universal Command Server for IBM i

Universal Copy

Copy from Local Windows to Remote IBM i via UCMD Manager

- Copy from Local Windows to Remote IBM i
  - Command Line Options
  - Components

Copy from Local Windows to Remote IBM i

The following figure illustrates the copying of a file initiated by Windows which copies a file from Windows to the first member of a file on IBM i.

```
ucmd -host sysName -userid userId -pwd password
-cmd "strucp tofile(mylib/readme)" < D:\tmp\README.txt
```

Using redirected stdin, UCMD Manager, running under Windows sends, transfers data to a UCMD Server running on the remote IBM i system, **sysName**. The UCMD Server on **sysName** invokes UCOPY to transfer the data to **`mylib/readme`**. **`mylib/readme`** file member ***FIRST** is used since TOMBR was not specified.

The command **STRUCP** is installed as part of UCMD Server on the IBM i system.

The file is copied as a text file since the default transfer mode is **text**.

Command Line Options

The command line options used in this example are:

Option	Description
-cmd	Remote command to execute on the IBM i.
-host	Directs the command to a computer with a host name of **sysName**.
-userid	IBM i user ID with which to execute the command.
-pwd	Password for the user ID.

Components

Universal Command Manager for Windows

Universal Command Server for IBM i

Universal Copy

Copy from Local IBM i to Remote Windows via UCMD Manager

- Copy from Local IBM i to Remote Windows via Universal Command Manager for IBM i
  - Command Line Options
  - Components

Copy from Local IBM i to Remote Windows via Universal Command Manager for IBM i

The following figures illustrate copying a file from a local IBM i system to a remote Windows system.

```
STRUCM CMD('ucopy > remotefile') HOST(dallas)
       USERID(joe) PWD(password) SINFILE(localfile)
```

The UCOPY utility running on the remote Windows system receives its standard in file from STRUCM via the Windows agent. The standard in of STRUCM is read from **localfile**, as specified by **SINFILE**.

The file is copied via UCMD as a text file, since the default transfer mode is **text**.

```
STRUCM CMD('ucopy -output c:\ntoutput.file') HOST(nthost) USERID(joe) PWD
(akkSdiq) SINFILE(library/inputfile) SINMBR(member)
```

The **ucopy** command receives its stdin file from STRUCM. The stdin of STRUCM is redirected from **SINFILE** and, optionally, **SINMBR** to stdout of **ucopy**, via the Windows agent.
UCOPY stdout is redirected to `c:\ntoutput.file` using the UCOPY **output** parameter.

The file is copied via UCMD as a text file, since the default transfer mode for standard files is text.

The command **ucopy** is installed as part of UCMD Server on the remote system.

Command Line Options

The command line options used are:

Option	Description
CMD	Remote command **`ucopy > file`** to execute. The **ucopy** program copies its stdin to its stdout.
HOST	Directs the command to a computer with a host name of **nthost**.
USERID	Remote user ID with which to execute the command.

PWD	Password for the user ID.
SINFILE [SINMBR]	Location from which the stdin file data is written.

Components

Universal Command Manager for IBM i

Universal Command Server for Windows

Universal Copy

Transferring Files to and from Remote Systems

- Transferring Files to and from Remote Systems
- Transfer Operation Components
  - Manager
  - Primary Server
  - Secondary Server

Transferring Files to and from Remote Systems

Universal Data Mover'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 Universal Data Mover transfer operation:

1. Manager
2. Primary server
3. Secondary server

The Manager can act as the primary server, depending on the type of transfer session: two-party or three-party (see Transfer Sessions).

The secondary server is always a separate and distinct component invoked via the Universal Broker.

Manager

The Universal Data Mover Manager processes commands using Universal Data Mover's scripting language. The Manager receives commands from the user through an interactive session, an external script file, or some combination of the two. Before the Manager can initiate any transfer operations, it must first establish a transfer session where it invokes the primary and secondary servers, which actually conduct the transfer operations.

Primary Server

When a transfer session is being established, the Universal Data Mover Manager invokes the primary server, which acts as the first endpoint in a transfer operation. In turn, the primary server invokes the secondary server, providing a single path of communication. The primary server also acts as a relay for the Manager, forwarding on any messages for the secondary server from the Manager. This single message pipeline reduces the number of connections needed for three-party transfers (see Three-Party Transfer Sessions).

Secondary Server

The secondary server acts as the second endpoint in a transfer operation. Data is transferred between primary and secondary servers, with either endpoint able to act as the source in a transfer operation.

Transfer Sessions

- Overview
  - Logical Names
  - Two-Party Transfer Sessions

- Three-Party Transfer Sessions
- Transfer Sessions (Illustrated)
- Detailed Information

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.)

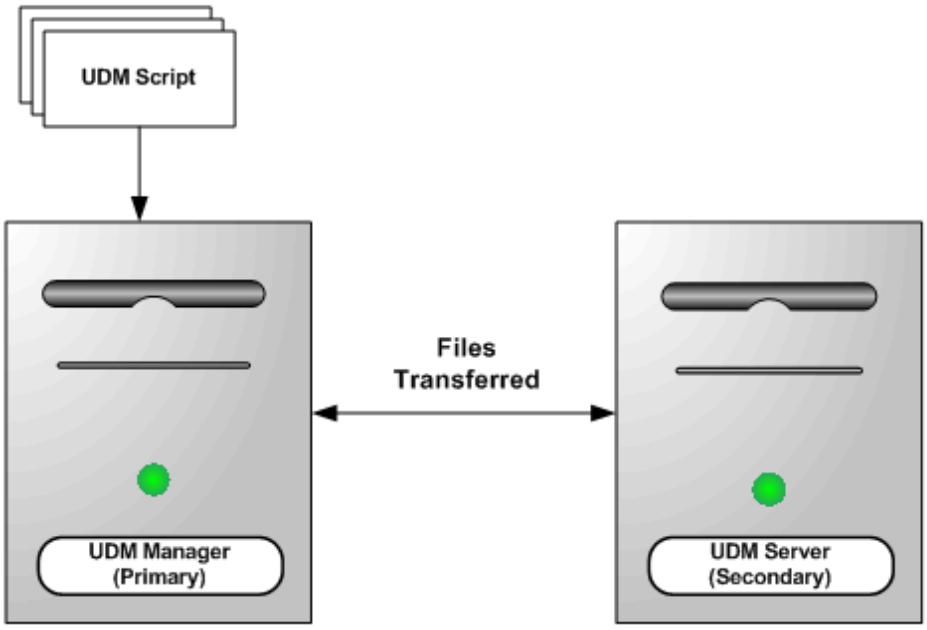
Three-Party Transfer Sessions

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.

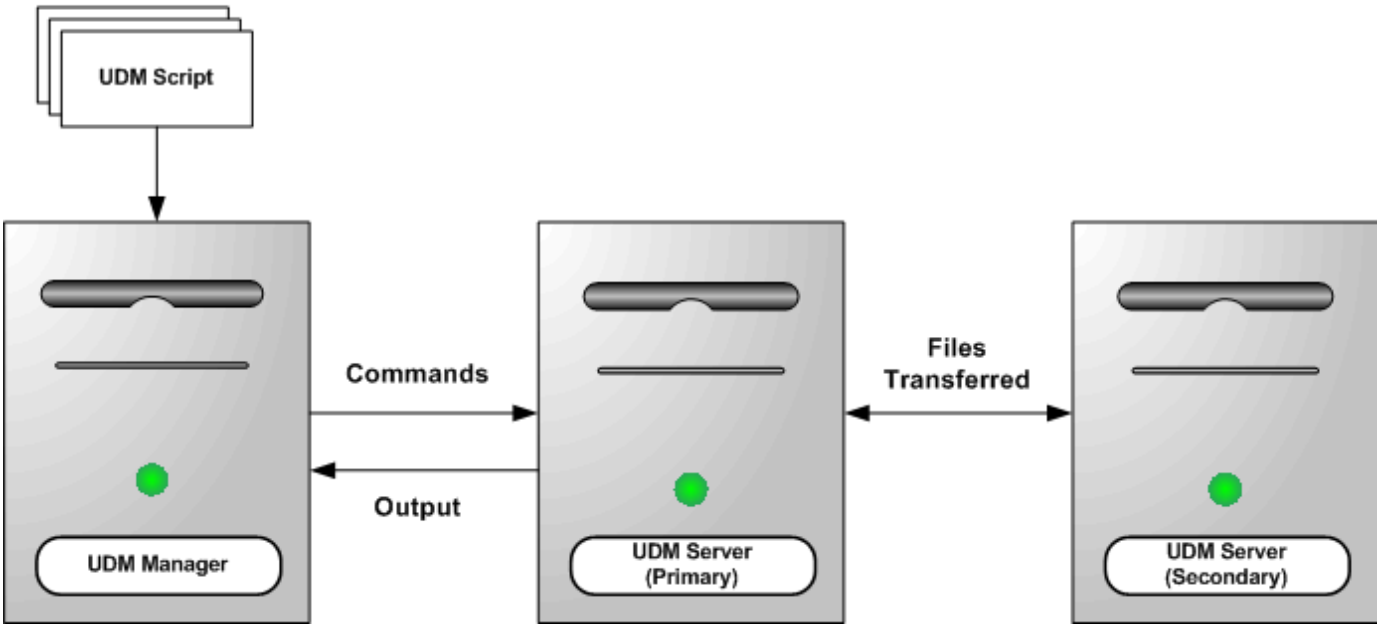
(See the following illustration.)

Transfer Sessions (Illustrated)

Two-Party Transfer



Three-Party Transfer



Detailed Information

For detailed information on opening transfer sessions and all UDM transfer operations, see Universal Data Mover Transfer Operations.

Transferring Files to and from Remote Systems - Examples

- Transferring Files Examples - z/OS
- Transferring Files Examples - Windows and UNIX
- Transferring Files Examples - IBM i

Transferring Files Examples - z/OS

- Copy a File to an Existing z/OS Sequential Data Set
- Copy a File to a New z/OS Sequential Data Set
- Copy a z/OS Sequential Data Set to a File
- Copy a Set of Files to an Existing z/OS Partitioned Data Set
- Copy a Set of Files to a New z/OS Partitioned Data Set

These examples illustrate two-party transfer sessions between z/OS and UNIX. As appropriate for the example being illustrated, there are versions for both the DSN and DD file systems.

They apply equally as well to the Windows and UNIX operating systems, with appropriate changes for the file system syntactical differences.

Transferring Files Examples - Windows and UNIX

- Simple File Copy to the Manager - Windows and UNIX
- Simple File Copy to the Server - Windows and UNIX
- Copy a Set of Files - Windows and UNIX

These examples illustrate two-party transfer sessions.

Each example illustrates a procedure that occurs under the default file system for that operating system.

See the list of z/OS and IBM i examples for file transfer examples that apply equally as well to the Windows operating systems.

Transferring Files Examples - IBM i

- Copy a File to an Existing IBM i File
- Copy an IBM i Data Physical File to a File
- Copy a Set of Files to an Existing Data Physical File
- Copy a File to a New IBM i Data Physical File
- Copy a File to a New IBM i Source Physical File
- Copy a Set of Files to a New Data Physical File on IBM i
- Copy Different Types of IBM i Files Using forfiles and $(_file.type)
- Invoke a Script from an IBM i Batch Job

Note
These examples reference the IBM i commands by their untagged names. If you are using commands with tagged names to run Universal Data Mover, substitute the tagged names for these untagged names. (For information on tagged names, see UCHGRLS (Change Release Tag) Program.)

These examples illustrate two-party transfer sessions between IBM i and UNIX. Each example illustrate a file transfer for the LIB file system.

They apply equally as well to the Windows and UNIX operating systems, with appropriate changes for the file system syntactical differences.

The first example, Copy a File to an Existing IBM i File, also includes a version specific to the HFS file system. For other examples similar to those used in the HFS file system, see #Transferring Files Examples - Windows and UNIX.

Copy a File to an Existing zOS Sequential Data Set

- Copy a File to an Existing z/OS Sequential Data Set
  - DD File System
  - DSN File System
  - Components

Copy a File to an Existing z/OS Sequential Data Set

These examples copy, in text mode, one sequential file to another. This is the simplest form of data transfer.

DD File System

```
//S1 EXEC UDMPRC
//APOUT DD DSN=APP.DATA.DAILY,DISP=SHR
//UNVSCR DD *
1 set _echo=yes
2 set _halton=warn
3 open unix=sol9 user=top098 pwd=p100m
4 filesys local=dd
5 cd unix=/opt/app/data
6 mode type=text
7 copy unix=data10.txt local=APOUT
8 quit /*
```

For this first z/OS example, the following is a line-by-line explanation:

1. Line 1 turns on command echo, which results in each command being written prior to processing.
2. Line 2 sets the error condition value on which script processing halts. Any error greater than or equal to warn halts script processing.
3. Line 3 opens a session between the local UDM Manager and a remote UDM server running on host **sol9**. The host **sol9** is given the logical name of **unix**. The open command also provides user credentials for the UDM server to verify and, if successfully verified, specifies the user ID with which the UDM server executes.
4. Line 4 changes the local file system from the default of DSN to DD. The file system type dictates the syntax and semantics of file specifications, such as in the copy command.
5. Line 5 changes the current directory of the UDM server **unix** running on host **sol9**.
6. Line 6 changes the transfer mode type from binary (the default) to text. Text mode transfers will translate between code pages (for example, ASCII and EBCDIC) and process the end-of-line characters.
7. Line 7 is the **copy** command that actually moves the data between systems. It copies file **data10.txt** on server **unix** to the local UDM Manager ddname APOUT. Recall that line 4 sets the local file system type to DD; hence, APOUT is referencing a ddname.
8. Line 8 executes the quit command, which closes all sessions and exits UDM with the highest exit code set.

DSN File System

```
//S1 EXEC UDMPRC
//UNVSCR DD *
1 set _echo=yes
2 set _halton=warn
3 open unix=sol9 user=top098 pwd=p100m
4 cd unix=/opt/app/data
5 mode type=text
6 attrib local createop=replace
7 copy unix=data10.txt local='app.data.daily'
8 quit /*
```

The DSN file system example is basically the same as the DD file system example, with these changes:

- Removal of the filesys command (line 4 in the DD file system example), since the default file system for the z/OS manager is DSN.
- Addition of line 6, which sets the local attribute **createop**. **createop** controls how a file is created. By default, its value is **new**, indicating that only new files are created and existing files are not written over (replaced). In this example, the value is being set to *replace*, which specifies that if the file exists, it should be replaced; otherwise, it is created.

Components

Universal Data Mover Manager for z/OS

Copy a File to a New zOS Sequential Data Set

```
```

- Copy a File to a New z/OS Sequential Data Set
  - DSN File System
  - Components

Copy a File to a New z/OS Sequential Data Set

This example copies, in text mode, a file from a remote UNIX system to a sequential data set on z/OS. The data set does not exist on z/OS; UDM is instructed to create it.

The data set is dynamically allocated based on the local UDM dynamic allocation attributes. UDM provides default attributes that can be changed to meet local requirements. The UDM defaults, as they are delivered, create a sequential, variable block record data set with a logical record length of 1024.

The sample below changes the record length to 256 in order to demonstrate how to set dynamic allocation attributes.

A DD file system sample is not provided, since creating a new data set with JCL is the same in UDM as it is in any batch application. There are no UDM specific requirements.

DSN File System

```
//S1     EXEC UDMPRC
//UNVSCR DD *
1 set _echo=yes _halton=warn
2 open unix=sol9 user=top098 pwd=p100m
3 cd unix=/opt/app/data
4 mode type=text
5 attrib local lrecl=256
6 copy data10.txt local='app.data.daily'
7 quit
/*
```

Note
All file names in the UNIX system must be within the eight-character range to be transferred successfully.

Almost all data set allocation attributes can be specified as UDM attributes, providing you with the ability to dynamically allocate any supported data set.

Care should be taken that conflicting allocation attributes are not specified. The results of the allocation should be 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
  - DD File System
  - DSN File System
  - Components

Copy a z/OS Sequential Data Set to a File

These examples copy, in text mode, a sequential data set on z/OS to a remote UNIX system.

Note
A text transfer, by default, does not trim spaces from the end of a record. If the data set being copied is a fixed record format, each record is padded with spaces so that the record length equals the logical record length of the data set. If you do not want the trailing spaces copied, they must be trimmed. Variable record formats do not normally have trailing spaces, so trimming normally is not required.

DD File System

```

```

```
//S1     EXEC UDMPRC
//APOUT  DD DSN=APP.DATA.DAILY,DISP=SHR
//UNVSCR DD *
1  set  _echo=yes
2  set  _halton=warn
3  open  unix=sol9 user=top098 pwd=p100m
4  filesys  local=dd
5  cd  unix=/opt/app/data
6  mode  type=text trim=yes
7  copy  local=apout unix=data10.txt
8  quit  /*
```

DSN File System

```
//S1     EXEC UDMPRC
//UNVSCR DD *
1 set _echo=yes
2 set _halton=warn
3 open unix=sol9 user=top098 pwd=p100m
4 cd unix=/opt/app/data
5 mode type=text trim=yes
6 copy local='app.data.daily' unix=data10.txt
7 quit
/*
```

Components

Universal Data Mover Manager for z/OS

Copy a Set of Files to an Existing zOS Partitioned Data Set

- Copy a Set of Files to an Existing z/OS Partitioned Data Set
  - DD File System
  - DSN File System
  - Components

Copy a Set of Files to an Existing z/OS Partitioned Data Set

These examples copy (in text mode, and using the * wildcard) multiple files with one **copy** command to an already allocated partitioned data set (PDS) on a z/OS system.

The file names used to create the member names in the destination PDS are the source file names.

However, note that file names on UNIX and Windows file systems often have a file extension as part of their name. A file extension is a suffix separated from the file's base name with a period (for example, BASE.TXT). The period in the file extension is not a valid character in PDS member names, so UDM must be instructed to remove the file extensions before copying them into the PDS.

The **truncext** attribute is used to instruct UDM to remove file name extensions from the source file prior to using the name as the destination member name.

This example assumes that the remote UNIX directory **/opt/app/data** contains the following list of files:

- **data001.txt**
- **data002.txt**
- **data003.txt**
- **data004.pr**
- **data005.pr**

The result of the copy operation will create the following members in PDS **APP.DATA.PDS**:

- **DATA001**
- **DATA002**
- **DATA003**

DD File System

```
//S1     EXEC UDMPRC
//APOUT  DD DSN=APP.DATA.PDS,DISP=SHR
//UNVSCR DD *
1  set  _echo=yes _halton=warn
2  open  unix=sol9 user=top098 pwd=p100m
3  filesys  local=dd
4  cd  unix=/opt/app/data
5  mode  type=text
6  attrib  local truncext=yes
7  copy  unix=*.txt local=apout
8  quit  /*
```

DSN File System

```
//S1     EXEC UDMPRC
//UNVSCR DD *
1 set _echo=yes _halton=warn
2 open unix=sol9 user=top098 pwd=p100m
3 cd unix=/opt/app/data
4 mode type=text
5 attrib local truncext=yes
6 copy unix=*.txt local='app.data.daily'
7 quit
/*
```

Components

Universal Data Mover Manager for z/OS

Copy a Set of Files to a New zOS Partitioned Data Set

- Copy a Set of Files to a New zOS Partitioned Data Set
  - DSN File System
  - Components

Copy a Set of Files to a New zOS Partitioned Data Set

This example copies, in text mode, a set of files from a remote UNIX system to a partitioned data set on z/OS. The data set does not exist on z/OS; UDM is instructed to create it.

The data set is dynamically allocated based on the local UDM dynamic allocation attributes. UDM provides default attributes that can be changed to meet local requirements. The UDM defaults as they are delivered create a sequential, variable block record data set with a logical record length of 1024.

This example changes the data set organization from sequential (PS) to partitioned (PO) and adjusts the data set's space allocation to space units of cylinders, primary space to 1, secondary space to 2, and directory blocks to 10.

DSN File System

```
//S1 EXEC UDMPRC
//UNVSCR DD *
1  set  _echo=yes _halton=warn
2  open  unix=sol9 user=top098 pwd=p100m
3  cd  unix=/opt/app/data
4  mode  type=text
5  attrib  local dsorg=po spaceunit=cyl primspace=1 secspace=2 +
6          dirblocks=10 truncext=yes
7  copy  unix=*.txt local='app.data.pds'
8  quit  /*
```

Note
Line 5 is continued onto line 6 with the line continuation character (+).

Components

Universal Data Mover Manager for z/OS

Simple File Copy to the Manager - Windows and UNIX

Simple File Copy to the Manager

This example copies, in text mode, one file to another. This is the simplest form of data transfer.

```
1  set _echo=yes
2  set _halton=warn
3  open unix=sol9 user=top098 pwd=p100m
4  cd unix=/opt/app/data
5  mode type=text
6  copy unix=data10.txt
7  quit
```

For this UNIX and Windows example, the following is a line-by-line explanation:

1. Line 1 turns on command echo, which results in each command being written prior to processing.
2. Line 2 sets error condition value on which script process halts. Any error equal to or greater than 4 halts script processing. A value of 4 effectively means halt on any error or warning.
3. Line 3 opens a session between the local UDM Manager and a remote UDM Server running on host **sol9**. The host **sol9** is given the a logical name of **unix**. The open command also provides user credentials for the UDM Server to verify and, if success verified, specifies the user ID with which the UDM Server executes.
4. Line 4 changes the current directory of the UDM server **unix** running on host **sol9**.
5. Line 5 changes the transfer mode type from binary (the default) to text. Text mode transfers will translate between code pages and process the end-of-line characters.
6. Line 6 is the copy command that actually moves the data between systems. It copies file **data10.txt** on server **unix** to the local UDM Manager as **data10.txt**.
7. Line 7 executes the quit command, which closes all sessions and exits UDM with the highest exit code set.

Components

Universal Data Mover Manager for Windows

Universal Data Mover Manager for UNIX

Simple File Copy to the Server - Windows and UNIX

Simple File Copy to the Server

This example copies, in text mode, a sequential data set on the UDM Manager machine to a remote UNIX system.

```
1 set _echo=yes
2 set _halton=warn
3 open unix=sol9 user=top098 pwd=p100m
4 cd unix=/opt/app/data
5 mode type=text
6 copy local=c:\data10.txt
7 quit
```

For this UNIX and Windows example, the following is a line-by-line explanation:

1. Line 1 turns on command echo, which results in each command being sent to stdout prior to processing.
2. Line 2 sets error condition value on which script process halts. Any error equal to or greater than 4 halts script processing. A value of 4 effectively means halt on any error or warning.

3. Line 3 opens a session between the local UDM Manager and a remote UDM server running on host **sol9**. The host **sol9** is given the a logical name of **unix**. The open command also provides user credentials for the UDM server to verify and, if success verified, specifies the user ID with which the UDM server executes.
4. Line 4 changes the current directory of the UDM server **unix** running on host **sol9**.
5. Line 5 changes the transfer mode type from binary (the default) to text. Text mode transfers will translate between code pages and process the end-of-line characters.
6. Line 6 is the copy command that actually moves the data between systems. It copies file **data10.txt** in the root directory on drive C of the Windows machine to the UNIX Server as **data10.txt**.
7. Line 7 executes the quit command, which closes all sessions and exits UDM with the highest exit code set.

Components

Universal Data Mover Manager for Windows

Universal Data Mover Manager for UNIX

Copy a Set of Files - Windows and UNIX

Copy a Set of Files

This example copies (in text mode, and using the * wildcard) multiple files with one copy.

It assumes that the remote UNIX directory **/opt/app/data** contains the following list of files:

- **data001.txt**
- **data002.txt**
- **data003.txt**
- **data004.pr**
- **data005.pr**

The following files will be created on the destination machine:

- **data001.txt**
- **data002.txt**
- **data003.txt**

The **truncext** attribute is used to instruct UDM to remove file name extensions from the source file prior to using the name as the destination member name.

```
1 set _echo=yes _halton=warn
2 open unix=sol9 user=top098 pwd=p100m
3 cd unix=/opt/app/data
4 mode type=text
5 attrib local truncext=yes
6 copy unix=*.txt
7 quit
```

Components

Universal Data Mover Manager for Windows

Universal Data Mover Manager for UNIX

Copy a File to an Existing IBM i File

- Copy a File to an Existing IBM i File
  - LIB File System
  - HFS File System
  - Components

Copy a File to an Existing IBM i File

These examples copy, in text mode, one sequential file to another. This is the simplest form of data transfer.

LIB File System

```
1 set _echo=yes
2 set _halton=warn
3 open unix=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
  - LIB File System
  - Components

Copy an IBM i Data Physical File to a File

This example copies, in text mode, a Data Physical File on IBM i to a remote UNIX system.

> **Note**
> A text transfer, by default, does not trim spaces from the end of a record. If the data set being copied is a fixed record format, each record is padded with spaces so that the record length equals the logical record length of the data set. If you do not want the trailing spaces copied, they must be trimmed.

LIB File System

```
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
  - LIB File System
  - Components

Copy a Set of Files to an Existing Data Physical File

This example copies (in text mode, and using the * wildcard) multiple files with one copy command to an already allocated Data Physical File on an IBM i system.

The file names used to create the member names in the destination Data Physical File are the source file names. However, note that file names on UNIX and Windows file systems often have a file extension as part of their name. A file extension is a suffix separated from the file's base name with a period (for example, BASE.TXT). Member names are limited to 10 characters on the IBM i system, so UDM must be instructed to remove the file extensions before copying them into the file.

The **truncext** attribute is used to instruct UDM to remove file name extensions from the source file prior to using the name as the destination member name.

This example assumes that the remote UNIX directory **/opt/app/data** contains the following list of files:

- **data001.txt**
- **data002.txt**
- **data003.txt**
- **data004.pr**
- **data005.pr**

The result of the copy operation will create the following members in Data Physical File APPDATA:

- **DATA001**
- **DATA002**
- **DATA003**

LIB File System

```
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
  - LIB File System
  - Components

Copy a File to a New IBM i Data Physical File

This example copies, in text mode, a file from a remote UNIX system to a data physical file on IBM i. The Data Physical File does not exist on IBM i; UDM is instructed to create it.

The file type created defaults to a Data Physical File. The Data Physical File is allocated based on the local UDM allocation attributes. UDM provides default attributes that can be changed to meet local requirements. The UDM defaults, as delivered, create a Data Physical File with a logical record length of 92 and maximum members of 1.

This example changes the record length to 80, and the maximum members to unlimited (*nomax*), in order to demonstrate how to set allocation attributes.

LIB File System

```
1 set _echo=yes _halton=warn
2 open unix=sol9 user=top098 pwd=p100m
3 cd unix=/opt/app/data
4 mode type=text
5 attrib local rcdlen=80 maxmbrs=nomax
6 copy unix=data10.txt local=MYLIB/APPDATA(DAILY)
7 quit
```

Almost all data set allocation attributes can be specified as UDM attributes giving you the ability to dynamically allocate any supported Data Physical File.

Care should be taken that conflicting allocation attributes are not specified. The results of the allocation should be checked to verify they meet your intentions. Although UDM checks attribute values, some values are provided by the system from sources that UDM cannot verify and can result in invalid or unintentional attribute combinations.

Components

Universal Data Mover Manager for IBM i

Copy a File to a New IBM i Source Physical File

- Copy a File to a New IBM i Source Physical File
  - LIB File System
  - Components

Copy a File to a New IBM i Source Physical File

This example copies, in text mode, a file from a remote UNIX system to a Source Physical File on IBM i. The Source Physical File does not exist on IBM i; UDM is instructed to create it.

The Source Physical File is allocated based on the local UDM allocation attributes. UDM provides default attributes that can be changed to meet local requirements. The UDM defaults, as delivered, create a Data Physical File with a logical record length of 92 and maximum members of 1.

This example changes the file type to **src** in order to demonstrate how to set allocation attributes.

LIB File System

```
1 set _echo=yes _halton=warn
2 open unix=sol9 user=top098 pwd=p100m
3 cd unix=/opt/app/data
4 mode type=text
5 attrib local filetype=src
6 copy unix=data10.txt local=MYLIB/APPDATA(DAILY)
7 quit
```

Almost all data set allocation attributes can be specified as UDM attributes giving you the ability to dynamically allocate any supported Data Physical File.

Care should be taken that conflicting allocation attributes are not specified. The results of the allocation should be checked to verify they meet your intentions. Although UDM checks attribute values, some values are provided by the system from sources that UDM cannot verify and may result in invalid or unintentional attribute combinations.

Components

Universal Data Mover Manager for IBM i

Copy a Set of Files to a New Data Physical File on IBM i

- Copy a Set of Files to a New Data Physical File on IBM i
  - LIB File System
  - Components

Copy a Set of Files to a New Data Physical File on IBM i

This example copies (in text mode, and using the * wildcard) a set of files from a remote UNIX system to a data physical file on IBM i. The data file does not exist on IBM i; UDM is instructed to create it.

The data set is allocated based on the local UDM allocation attributes. UDM provides default attributes that can be changed to meet local requirements. The UDM defaults, as delivered, create a data physical file with a logical record length of 92 and maximum members of 1.

This example changes the record length to 80 and the maximum members to unlimited (*nomax*).

LIB File System

```
1 set _echo=yes _halton=warn
2 open unix=sol9 user=top098 pwd=p100m
3 cd unix=/opt/app/data
4 mode type=text
5 attrib local maxmbrs=nomax rcdlen=80 truncext=yes
6 copy unix=*.txt local=MYLIB/APPDATA
7 quit
```

Components

Universal Data Mover Manager for IBM i

Copy Different Types of IBM i Files Using forfiles and $(_file.type)

- Copy Different Types of IBM i Files using forfiles and $(_file.type)
  - LIB File System
  - Components

Copy Different Types of IBM i Files using forfiles and $(_file.type)

Physical files are considered directories in UDM because they contain 1+ member. Save files are considered files because they do not contain any members. The forfiles statement and the variable **$(_file.type)** allow you to do a wildcard copy on both save and physical files in the LIB file system.

This example copies a mix of files (Save and Physical) from an IBM i system in a single operation, using the forfiles statement and the **$(_file.type)** variable attribute.

LIB File System

```
forfiles src=MYLIB/*
if $(_file.type) EQ directory
copy src=$(_path)\(*)
elsecopy src=$(_path)
end
end
```

Components

Universal Data Mover Manager for IBM i

Invoke a Script from an IBM i Batch Job

- Invoke a Script from an IBM i Batch Job
  - LIB file system
  - Components

Invoke a Script from an IBM i Batch Job

To invoke a script included as an inline file in a database job, the call must specify ***FIRST** as the database member name.

The following example illustrates both:

- Invocation of an inline script, **CALLME**, using the STRUDM command from a database job.
- Invocation of an inline script, **CALL1**, using the CALL command from a database job.

LIB file system

```
//BCHJOB JOB(testcall) ENDSEV(10) OUTQ(mytest/UDMOUTQ) LOGCLPGM(*YES) LOG(2
20 \*SECLVL) MSGQ(*USRPRF)
//DATA FILE(CALL1) ENDCHAR(ENDDATAFILE)
print msg="I made it to call1 - an inline file"
ENDDATAFILE
//DATA FILE(CALLME) ENDCHAR(ENDDATAFILE)
OPEN S=AS400V5 USER=qatest PWD=****\* PORT=4311
CALL CALL1(*FIRST)
CLOSE
ENDDATAFILE
STRUDM SCRFILE(CALLME)
//ENDBCHJOB
```

Components

Universal Data Mover Manager for IBM i

Encryption

- Encryption
- Encrypting Files
- Transferring Encrypted Files between Servers
  - Security Considerations
- Universal Broker Key Store
- Additional Information

Encryption

Universal Agent programs have the ability to read command line options contained in command files. Command files that contain private information must be protected by using local file system security. This ensures that only authorized accounts have read access.

The Universal Encrypt (**UENCRYPT**) utility adds an additional layer of security by encrypting the contents of command files into an unintelligible format.

Although all command line options can be encrypted with Universal Encrypt, most organizations use it to encrypt and store authentication credentials such as user ID and/or password.

An encrypted command file can be decrypted only by Stonebranch product programs. No decrypt command is provided to decrypt the command file.



Note
Universal Encrypt should not be used as a replacement for file system security.

Encrypting Files

Files do not have to be encrypted on the same platform or server on which they will be used. They can be encrypted on any platform or server and then transferred. This means that applications development, platform administrators, and security administrators can encrypt passwords in their own environments.

Universal Encrypt encrypts files with either:

- 56-bit DES
- 256-bit AES

Universal Encrypt reads an unencrypted file from its standard input and writes the encrypted version to its standard output.

Encrypted files are text files and contain comments that can be edited if required. Lines within the encrypted file that start with the # character are comments. Default comments are created with the following information:

- Date of encryption.
- Userid that encrypted the file.
- System on which the file was encrypted.
- Version of Universal Encrypt used.
- Level of encryption used.

Transferring Encrypted Files between Servers

Files encrypted via Universal Encrypt are text files.

You can transfer them between servers, using FTP or similar tools, in text mode. You also can email them between like systems (for example, Windows to Windows).

Security Considerations

For production implementations, thought should be given to the location and security of encrypted files containing passwords. Consider who needs access to create, update, and use these files.

Many implementations are centralized around an enterprise scheduling solution. In this case, the encrypted files are often secured in such a way that only the enterprise scheduler is able to access them.

There are additional layers of security available to Universal Agent, such as Universal Access Control List and X.509 Certificates. These can be further used to ensure that access to servers is properly controlled.

Universal Broker Key Store

During installation, you can request the generation of an encryption key, which is stored in a Universal Broker key store.

If a Universal Agent component wants to use this encryption key, it requests it from the Universal Broker.

For detailed information on encryption keys and the key store, see Universal Broker Key Store.

Additional Information

The following pages provide additional detailed information for Encryption:

- Encryption - Examples

Encryption - Examples

Examples

The following pages provide examples of how to use Universal Encrypt to encrypt a command file (and how to use the encrypted file). Each example will encrypt a case sensitive password using AES 256 encryption.

Links to detailed technical information on appropriate Universal Agent components are provided for each example.

- Creating Encrypted Command File on z/OS
- Using Encrypted Command File on z/OS
- Creating Encrypted Command File on Windows
- Using Encrypted Command File on Windows
- Creating Encrypted Command File on UNIX
- Using Encrypted Command File on UNIX
- Creating Encrypted Command File on IBM i
- Using Encrypted Command File on IBM i
- Creating Encrypted Command File on HP NonStop

Creating Encrypted Command File - zOS

- Creating Encrypted Command File for z/OS
  - Command File
  - JCL
  - Contents of Encrypted File
  - Components

Creating Encrypted Command File for z/OS

Command File

In this example, a Universal Command command file named **MY.CLEAR.CMDFILE** contains the following data:

```
-userid T02JAH1 -pwd thames
```

Command File Options

The command file options used in this example are:

Option	Description
-userid	User ID or account with which to execute the remote command.
-pwd	Password associated with -userid.

Note
If you are creating an encrypted file for use with the Universal Controller CLI (Command Line Interface), you must use the CLI command line switches to specify the user ID and password:

- -p (password)
- -u (user ID)

JCL

The following JCL encrypts the command file allocated to ddname **UNVIN** using AES encryption and an encryption key **MYKEY123**:

```
//UENCRYPT EXEC PGM=UENCRYPT
//STEPLIB  DD  DISP=SHR,DSN=UNV.SUNVLOAD
//SYSPRINT DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
//UNVIN    DD  DISP=SHR,MY.CLEAR.CMDFILE
//UNVOUT   DD  DISP=SHR,MY.ENCRYPT.CMDFILE
//SYSIN    DD  *
-key MYKEY123 -aes YES
/*
```

The resulting encrypted command file is written to ddname **UNVOUT**.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-key	Encryption key used by the encryption algorithm.
-aes	Specification for whether or not AES encryption is used.

Contents of Encrypted File

The figure below illustrates the contents of **MY.ENCRYPT.CMDFILE**.

```
# Universal Encrypt
#   Date . . . . . : Thu Nov  3 07:29:03 2011
#   User . . . . . : T02JAH1
#   Host . . . . . : hosta.acme.com
#   Program  . . . : uencrypt 3.2.0 Level 5 Release Build 130
#   Encryption . . : AES 256-bit

1F7DAF62583C813EA874CA168FF626C348F7BF171477D380D9A2FFFED33C539B71B4206EA5021
F
D92CDFDD931C3B88B9CD711A4693EFE6B49FAE9431E9C946F7F35C9B4C31335BFB3F97F0686EF
F
37068245A6B58CBFE2ADE32997A132C4114AC52CD615B2E7E8672ED0BF9867CA13B1
```

This encrypted command file can now be used by any Universal Agent command on any platform by specifying the encryption key **MYKEY123**.

Components

Universal Encrypt

Using Encrypted Command File - zOS

Using Encrypted Command File on z/OS

For z/OS, the Universal Command Manager -encryptedfile option specifies the ddname in the JCL that references the location of the Uencrypted file.

```
//UCM#000  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*
//         JCLLIB ORDER=#HLQ.UNV.SUNVSAMP
//*
//STEP1    EXEC UCMDPRC
//UENCRYPT DD  DISP=SHR,DSN=TEST.UENFILES(TESTPWD)
//COMMANDS DD  *
 DIR
//SYSIN    DD  *
 -host                10.252.2.232
 -userid              "testid"
 -encryptedfile       UENCRYPT
 -script              COMMANDS
```

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-host	List of one or more hosts upon which a command can run.
-userid	User ID or account with which to execute the remote command.
-encryptedfile	Encrypted command file.

-script	Local script file to execute on the remote system.

Components

Universal Command Manager for z/OS

Universal Encrypt

Creating Encrypted Command File - Windows

- Creating Encrypted Command File for Windows
  - Command File
  - Encryption Command
  - Contents of Encrypted File
  - Components

Creating Encrypted Command File for Windows

Command File

In this example, a Universal Command command file named **`cmdfile.txt`** contains the following data:

```
-userid T02JAH1 -pwd thames
```

Command File Options

The command file options used in this example are:

Option	Description
-userid	User ID or account with which to execute the remote command.
-pwd	Password associated with -userid.

Note
If you are creating an encrypted file for use with the Universal Controller CLI (Command Line Interface), you must use the CLI command line switches to specify the user ID and password:

- -p (password)
- -u (user ID)

Encryption Command

The following command encrypts the command file using AES encryption with an encryption key **MYKEY123**.

```
uencrypt -key MYKEY123 -aes yes <cmdfile.txt> encfile.txt
```

Note
The name of the file containing the text to be encrypted (in this case, cmdfile.txt} must be enclosed in less than (<) and greater then (>) characters.

The resulting encrypted command file, in this case, is encfile.txt.

Command Line Options

The command line options used in this example are:

Option	Description
-key	Encryption key used by the encryption algorithm.
-aes	Specification for whether or not AES encryption is used.

Contents of Encrypted File

The following figure illustrates the contents of **`encfile.txt`**.

```
# Universal Encrypt
#   Date . . . . . : Thu Nov  3 07:29:03 2011
#   User . . . . . : T02JAH1
#   Host . . . . . : hosta.acme.com
#   Program  . . . : uencrypt 3.2.0 Level 5 Release Build 130
#   Encryption . . : AES 256-bit

1F7DAF62583C813EA874CA168FF626C348F7BF171477D380D9A2FFFED33C539B71B4206EA502
1FD92CDFDD931C3B88B9CD711A4693EFE6B49FAE9431E9C946F7F35C9B4C31335BFB3F97F068
6EFF37068245A6B58CBFE2ADE32997A132C4114AC52CD615B2E7E8672ED0BF9867CA13B1
```

This encrypted command file now can be used by any Universal Agent command, on any operating system, by specifying the encryption key **MYKEY123**.

Components

Universal Command Manager for Windows

Universal Encrypt

Using Encrypted Command File - Windows

Using Encrypted Command File on Windows

For Windows, the Universal Command Manager -encryptedfile option specifies the location of the Uencrypted file.

```
ucmd -host 10.252.2.232 -userid testid -encryptedfile c:
\Universal\Encrypted\enc.txt -cmd "dir"
```

Command Line Options

The Command options used in this example are:

Option	Description
-host	List of one or more hosts upon which a command can run.
-userid	User ID or account with which to execute the remote command.
-encryptedfile	Encrypted command file.
-cmd	Remote command to execute.

Components

Universal Command Manager for Windows

Universal Encrypt

Creating Encrypted Command File - UNIX

- Creating Encrypted Command File for UNIX
  - Command File
  - Encryption Command
  - Contents of Encrypted File
  - Components

Creating Encrypted Command File for UNIX

Command File

In this example, a Universal Command command file named **`cmdfile.txt`** contains the following data:

```
-userid T02JAH1 -pwd thames
```

Command File Options

The command file options used in this example are:

Option	Description
-userid	User ID or account with which to execute the remote command.
-pwd	Password associated with -userid.

Note
If you are creating an encrypted file for use with the Universal Controller CLI (Command Line Interface), you must use the CLI command line switches to specify the user ID and password:

- -p (password)
- -u (user ID)

Encryption Command

The following command encrypts the command file using AES encryption with an encryption key **MYKEY123**.

```
uencrypt -key MYKEY123 -aes yes <cmdfile.txt> encfile.txt
```

Note
The name of the file containing the text to be encrypted (in this case, cmdfile.txt} must be enclosed in less than (<) and greater then (>) characters.

The resulting encrypted command file, in this case, is encfile.txt.

Command Line Options

The command line options used in this example are:

Option	Description
-key	Encryption key used by the encryption algorithm.
-aes	Specification for whether or not AES encryption is used.

Contents of Encrypted File

The following figure illustrates the contents of **`encfile.txt`**.

```
# Universal Encrypt
#   Date . . . . . : Thu Nov  3 07:29:03 2011
#   User . . . . . : T02JAH1
#   Host . . . . . : hosta.acme.com
#   Program  . . . : uencrypt 3.2.0 Level 5 Release Build 130
#   Encryption . . : AES 256-bit

1F7DAF62583C813EA874CA168FF626C348F7BF171477D380D9A2FFFED33C539B71B4206EA502
1FD92CDFDD931C3B88B9CD711A4693EFE6B49FAE9431E9C946F7F35C9B4C31335BFB3F97F068
6EFF37068245A6B58CBFE2ADE32997A132C4114AC52CD615B2E7E8672ED0BF9867CA13B1
```

This encrypted command file now can be used by any Universal Agent command, on any operating system, by specifying the encryption key **MYKEY123**.

Components

Universal Command Manager for UNIX

Universal Encrypt

Using Encrypted Command File - UNIX

Using Encrypted Command File on UNIX

For the UNIX, the Universal Command Manager -encryptedfile option specifies the location of the Uencrypted file.

```
/opt/universal/bin/ucmd -host 10.252.2.232 -userid testid \
-encryptedfile /universal/encrypted/encfile.txt -cmd "dir"
```

Command Line Options

The Command options used in this example are:

Option	Description
-host	List of one or more hosts upon which a command can run.
-userid	User ID or account with which to execute the remote command.
-encryptedfile	Encrypted command file.
-cmd	Remote command to execute.

Components

Universal Command Manager for UNIX

Universal Encrypt

Creating Encrypted Command File - IBM i

- Creating Encrypted Command File for IBM i
  - Command File
  - Encryption Command
  - Contents of Encrypted File
  - Components

Creating Encrypted Command File for IBM i

Command File

In this example, a Universal Command command file named **`MYLIB/QTXTSRC(TESTLOGIN)`** contains the following data:

```
-userid T02JAH1 -pwd tz74gan
```

Command File Options

The command file options used in this example are:

Option	Description
-userid	User ID or account with which to execute the remote command.
-pwd	Password associated with -userid.

Encryption Command

The following command encrypts the command file using non-AES encryption with an encryption key **MYKEY123** for default codepage IBM1047.

```
STRUEN INFILE(MYLIB/QTXTSRC) INMBR(TESTLOGIN) OUTFILE(MYLIB/ENCRYPTEDF)
OUTMBR(ENCRYPTEDF) KEY(MYKEY123)
```

The resulting encrypted command file is written to file **ENCRYPTEDF** in **MYLIB** library.

Command Line Options

The command line options used in this example are:

Option	Description
INFILE	Input file that is to be encrypted.
INMBR	Location of data in the input file that is to be encrypted.
OUTFILE	File to which the encrypted input file is written.
OUTMBR	Location of data in the file to which the encrypted input file is written.
KEY	Encryption key used by the encryption algorithm.

Contents of Encrypted File

The figure below illustrates the contents of **`MYLIB/ENCRYPTEDF(ENCRYPTEDF)`**.

```
# Universal Encrypt
# Created on Wed Feb 22 18:43:51 2011
# Created by uencrypt 3.2.0 Level 0

9ACB96416816600CB9D24C9072D80C11768B93CB0E79B944EC37D3495097AD793F97399220C9B
B
472DF1E04F5BA8909BCA6C8C72DFD3B706487B1713E6F73F5A0539F17076DEF6D14083EF6E702
3
158526E70BE3AF688579805DCAC0CFF1EB6A
```

This encrypted file now can be used as command file input for a Universal Agent command on any platform that uses the encryption key **MYKEY123**.

Components

Universal Command Manager for IBM i

Universal Encrypt

Using Encrypted Command File - IBM i

Using Encrypted Command File on IBM i

For IBM i, the Universal Command Manager ECMFILE / ECMMBR option specifies the location of the Uencrypted file.

```
STRUCM HOST('10.252.2.232') USERID(testid) ECMFILE(UNIVERSAL/ENCRYPTED)
ECMMBR(TETSPWD) CMD('DIR')
```

Command Line Options

The command line options used in this example are:

Option	Description

HOST	List of one or more hosts upon which a command can run.
USERID	User ID or account with which to execute the remote command.
ECMFILE	Encrypted command file.
ECMMBR	Location of encrypted data in encrypted command file.
CMD	Remote command to execute.

Components

Universal Command Manager for IBM i

Universal Encrypt

Configuration Management for Universal Agent

Overview

Configuration consists of specifying options that control component behavior and resource allocation.

- An example of configurable component behavior is whether or not data transferred over the network is compressed.
- An example of configurable resource allocation is the directory location in which the product creates its log files.

Configuration can be done either by:

- Setting default options and preferences for all executions of a component.
- Setting options and preferences for a single execution of a component.

Each option is comprised of a pre-defined parameter, which identifies the option, and one or more values. The format of the parameter depends on the method being used to specify the option.

Although there are many configurable component options, components are - in general - designed to require minimal configuration and administration. The default options will work very well in most environments. When local requirements do require a change in component configuration, there are multiple methods available to configure the components in order to meet your needs.

Detailed Information

The following pages provide detailed information for Configuration Management:

- Configuration Methods
- Remote Configuration
- Universal Configuration Manager
- Configuration Refresh
- Refreshing via Universal Control Examples
- Merging Configuration Options
- Configuration Options

Configuration Methods

- Configuration Methods
  - Universal Broker / Servers Configuration Method
  - z/OS Platform

Configuration Methods

All components provide a consistent and flexible method of configuration. An operating system's native configuration methods, such as configuration files, are utilized in order to integrate with existing system management policies and procedures for the platform.

Depending on the specific Universal Agent component, and the operating system on which it is being run, component configuration is performed by one or more methods.

These configuration methods, in their order of precedence, are:

1. Command Line

2. Command File
3. Environment Variables
4. Configuration File

The command line, command file, and environment variables methods let you set configuration options and preferences for a single execution of a component.

The configuration file method lets you set default configuration options and preferences for all executions of a component.

This order of precedence means that an option specified on the command line overrides the same option specified in a command file, which overrides the same option specified with an environment variable, which overrides the same option specified in a configuration file.

> **Note**
> For security reasons, not all options can be overridden.

Universal Broker / Servers Configuration Method

Universal Broker, and all Universal Agent servers, are configurable only by modifying their configuration files (see Configuration File). They are not configurable via command line, command file, or environmental variables.

z/OS Platform

On the z/OS platform, configuration can utilize z/OS system symbols as part of the configuration value. Each system symbol is resolved when the value is first read by a component.

z/OS System symbols may be used in some of the configuration methods as follows:

- Command line or command file options prefixed with a plus (+) character instead of a dash (-) result in system symbols in the option value being resolved.
- System symbols are not supported in environment variables.
- System symbols are always resolved in configuration file values.

System symbols start with the ampersand character (**&**) and end with a period (**.**). For example, the **&SYSNAME.** symbol specified in the Universal Broker UNIX_DB_DATA_SET option is **"UNV.&SYSNAME..UNVDB"**. The variable **"&SYSNAME."** will be replaced with the symbol value.

The z/OS system symbols that are defined on z/OS can be displayed with the MVS system command DISPLAY SYMBOLS.

Command Line

Command line options affect one instance of a program execution. Each time that you execute a program, command line options let you tailor the behavior of the program to meet the specific needs for that execution.

Command line options are the highest in order of precedence of all the Configuration Methods. They override the options specified using all other configuration methods, except where indicated.

Each command line options consist of:

- Parameter (name of the option)
- Value (pre-defined or user-defined value of the option)

The command line syntax depends, in part, on the operating system, as noted below.

A value may or may not be case-sensitive, depending on what it is specifying. For example, if a value is either **yes** or **no**, it is not case-sensitive. It could be specified as **YES**, **Yes**, or **yes**. However, if a value specifies a directory name or file name, it would be case-sensitive if the operating system's file system is case-sensitive.

If an option is specified more than once on the command line, the last instance of the option specified is used.

z/OS	Command line options are specified in the JCL EXEC statement PARM keyword or on the SYSIN ddname. The PARM keyword is used to pass command line options to the program being executed with the EXEC statement. Command line options are prefixed with a dash (-) character or a plus (+) character. The plus character indicates that system symbols found in the value are resolved to their defined value before the value is processed by the Universal Agent component. For many options, there are two different forms in which they can be specified: • Short form: one case-sensitive character • Long form: two or more case-insensitive characters

	The parameter and value must be separated by at least one space.
	
	Example command line options specified in the PARM value:
	
	Short form:
	`PARM='-l INFO -G yes'`
	
	Long form:
	`PARM='-LEVEL INFO -LOGIN YES'`
	
	As noted above, z/OS command line options also can be specified on the SYSIN ddname. This is the easiest and least restrictive place to specify options, since the PARM values are limited in length. The options specified in the SYSIN ddname have the same syntax. Options can be specified on one line or multiple lines. The data set or inline data allocated to the SYSIN ddname cannot have line numbers in the last 8 columns (that is, all columns of the records are used as input).

UNIX, Windows, HP NonStop	Command line options are prefixed with a dash (**-**) character, and alternatively on Windows, the slash (**/**) character.
	
	For many options, there are two different forms in which they can be specified:
	
	• Short form: one case-sensitive character.
	• Long form: two or more case-insensitive characters.
	
	The parameter and value must be separated by at least one space or tab character.
	
	Example command line options:
	
	Short form:
	`-l info -G yes`
	
	Long form:
	`-level info -login yes`
	`-LEVEL info -LoGiN YES`

IBM i	Command line options use the native conventions for Command Language (CL) commands. The option name is specified as a CL parameter with its value enclosed in parentheses. Example command line options: `MSGLEVEL(INFO) COMPRESS(*YES)` All Universal Agent components provide IBM i-style command panels. The panels are accessed by entering the command name on the command line and pressing the F4 (PROMPT) key.

Command File

The command file contains command line options specified in a file. The command file enables you to save common command line options in permanent storage and reference them as needed.

The command file is the second to highest in the precedence order, after command line options (see Configuration Methods).

Individual command line options can be specified on one or multiple lines. Blank lines are ignored. Lines starting with the hash (**#**) character are ignored and can be used for comments.

The command file can be encrypted if it is necessary to secure the contents (see Universal Encrypt).

> **Note**
> If the contents of the file contain sensitive material, the operating system's native file and user security facilities should be used in addition to the file encryption provided by Universal Agent.

In order to use a command file, either of the following is used:

- COMMAND_FILE_PLAIN option is used to specify the command file name.
- COMMAND_FILE_ENCRYPTED option is used to specify the encrypted command file name.

Configuration Methods - Environment Variables

Environment Variables

Environment variables, like command line options, allow options to be specified for one instance of a program execution. Each time that you execute a program, environment variables allow you to tailor the behavior of the program to meet the specific needs for that execution.

Environment variables are the third to highest in the precedence order, after command file options (see Configuration Methods).

Each operating system has its own unique method of setting environment variables.

All environment variables used by Universal Agent are upper case and are prefixed with a product identifier consisting of three or four characters. The product sections specify the value of the environment variables. Values are case-sensitive.

z/OS	Environment variables in z/OS are specified in the JCL EXEC statement PARM keyword. Environment variables are part of the IBM Language Environment (LE) and as such are specified as LE runtime options. The PARM value is divided into LE options and application options by a slash (**/**) character. Options to the left of the slash are LE options and options to the right are application options. Example of setting an environment variable (set option UCMDLEVEL to a value of INFO): `PARM='ENVAR("UCMDLEVEL=INFO")/'`

UNIX	Environment variables in UNIX are defined as part of the shell environment. As such, shell commands are used to set environment variables. The environment variable must be exported to be used be a called program.
	
	Example of setting an environment variable (set option UCMDLEVEL to a value of INFO in a bourne, bash, or korn shell):
	
	`UCMDLEVEL=INFO`
	`export UCMDLEVEL`

Windows	Environment variables in Windows are defined as part of the Windows console command environment. As such, console commands are used to set environment variables. Example of setting an environment variable (set option UCMDLEVEL to a value of INFO): `SET UCMDLEVEL=INFO`
IBMi	Environment variables in IBM i are defined with Command Language (CL) commands for the current job environment. Example of setting an environment variable (set option UCMDLEVEL to a value of INFO): `ADDENVVAR ENVVAR(UCMDLEVEL) VALUE(INFO)`

Configuration Methods - Configuration File

- Configuration File
- Configuration File Syntax

Configuration File

Configuration files are used to specify system-wide configuration values. This method is last in the order of precedence; that is, configuration file option values can be overridden by every other method of configuration (see Configuration Methods).

For most Universal Agent components, some options can be specified only in a configuration file, while other options can be overridden by individual command executions. The Reference Guide for each component identifies these options.

Universal Broker maintains the for all Universal Agent components, including itself. The components do not read their configuration files themselves (except for Universal Enterprise Controller, which does read its own configuration file).

At initial start-up, Universal Broker reads the configuration files of all components and places the configuration data in Universal Broker memory. When a component starts, it first registers with its local Universal Broker. As part of the registration process, Universal Broker sends that component's configuration data to the component. Thereafter, if a configuration file is modified, Universal Broker must be refreshed. This directs Universal Broker to re-read all component configuration files and update the configuration data in memory (see Configuration Refresh).

Universal Broker can operate in managed or unmanaged mode:

- In unmanaged mode, the configuration information for the various Universal Agent components can be modified either:
  - Locally (either by editing the configuration files or, on Windows systems, via the Universal Configuration Manager).
  - Remotely, via the Universal Enterprise Controller I-Management Console application.
- In managed mode, the configuration information for the various Universal Agent components is "locked down" and can be modified or viewed only via the I-Management Console.

(For information on unmanaged and managed modes, see Remote Configuration).

z/OS	Configuration files are members of a PDSE. The data set record format is fixed or fixed block with a record length of 80. No line numbers can exist in columns 72-80. All 80 columns are processed as data. All configuration files are installed in the **UNVCONF** library. See #Configuration File Syntax, below, for the configuration file syntax.
UNIX	Configuration files are regular text files on UNIX. The files can be edited with a text editor. See #Configuration File Syntax, below, for the configuration file syntax. Universal Broker searches for the configuration files in a fixed list of directories. The Broker will use the first configuration file that it finds in its search. The directories are listed below in the order they are searched. • /etc/opt/universal • /etc/universal (installation default) • /etc/stonebranch (obsolete as of version 2.2.0) • /etc • /usr/etc/universal • /usr/etc/stonebranch (obsolete as of version 2.2.0) • /usr/etc
Windows	Although configuration files can be edited with any text editor (for example, Notepad), the Universal Configuration Manager application, accessible via the Control Panel, is the recommended way to set configuration options. Universal Configuration Manager provides a graphical interface and context-sensitive help, and helps protect the integrity of the configuration file by validating all changes to configuration option values.
IBMi	The configuration files on IBM i are stored in a source physical file named UNVCONF in the UNVPRD520 library. The files can be edited with a text editor. See #Configuration File Syntax, below, for the configuration file syntax.
HPNonStop	The configuration files on HP NonStop are stored as EDIT files, file code 101, within the **$SYSTEM.UNVCONF** subvolume. The files can be edited with the EDIT editor. See #Configuration File Syntax, below, for the configuration file syntax.

Configuration File Syntax

Configuration files are text files that can be edited with any available text editor.

The following rules apply for configuration file syntax:

- Options are specified in a keyword / value format.
- Keywords can start in any column.
- Keywords must be separated from values by at least one space or tab character.
- Keywords are not case sensitive.
- Keywords cannot contain spaces or tabs.
- Values can contain spaces and tabs, but if they do, they must be enclosed in single (**'**) or double (**"**) quotation marks. Repeat the enclosing characters to include them as part of the value.

- Values case sensitivity depends on the value being specified. For example:
  - Directory and file names are case sensitive.
  - Pre-defined values (such as **yes** and **no**) are not case sensitive.
- Each keyword / value pair must be on one line.
- Characters after the value are ignored.
- Newline characters are not permitted in a value.
- Values can be continued from one line to the next either by ending the line with a:
  - Plus (**+**) character, to remove all intervening spaces.
  - Minus (**-**) character, to preserve all intervening spaces between the end of the line being continued and the beginning of the continuing line.
- Ensure that the line continuation character is the last character on a line.
- Comment lines start with a hash (#) character.
- Blank lines are ignored.

Note
If an option is specified more than once in a configuration file, the last instance is used.

Remote Configuration

- Remote Configuration
- Unmanaged Mode
- Managed Mode
  - Selecting Managed Mode
- Universal Broker Start-up
  - Start-up in Unmanaged Mode
  - Start-up in Managed Mode

Remote Configuration

Universal Agent components can be configured remotely by Universal Enterprise Controller via the I-Management Console client application, and can be "locked down" so that they *only* can be remotely configured.

I-Management Console instructs the Universal Broker of a remote Agent to modify the configurations of all Universal Agent components managed by that Universal Broker.

Universal Broker supports remote configuration in either of two modes:

1. Unmanaged Mode
2. Managed Mode

Unmanaged Mode

Unmanaged mode is the default mode of operations for Universal Broker. It allows a Universal Broker - and the Universal Agent components managed by that Universal Broker - to be configured either:

- Locally, by editing configuration files.
- Remotely, via I-Management Console.

The system administrator for the machine on which an Agent resides can use any text editor to modify the configuration files of the various local Universal Agent components.

Via I-Management Console, selected users can modify all configurations of any Agent, including the local Agent. I-Management Console sends the modified data to the Universal Broker of that agent, which Universal Broker then uses to update the appropriate configuration files.

If I-Management Console sends modifications for a Universal Broker configuration, Universal Broker validates the modified data before it accepts it. If the data fails validation, Universal Broker does not update its configuration file.

If I-Management Console sends modification to the configuration of any other Universal Agent component, the Universal Broker updates the appropriate configuration file. The component will use this new configuration at its next invocation.

Note
If errors or invalid configuration values are updated via I-Management Console for a component other than Universal Broker, the component may not run successfully until the configuration has be corrected.

Managed Mode

When a Universal Broker is operating in managed mode, the configuration information for all Universal Agent components managed by that Universal Broker is "locked down." Universal Broker stores the information in a database file located within its specified spool directory. The information can be modified only via I-Management Console.

From this point on, Universal Broker uses the database file - not the configuration files - to access configuration information. Any configuration changes made to the components - via I-Management Console - are placed in the database file. Therefore, as long as Universal Broker stays in managed mode, the configuration files may no longer contain current or valid configuration information.

If managed mode is de-selected for the Universal Broker, it reads the database file where it stored the configuration information. Universal Broker uses this information to create and/or update configuration files for the components.

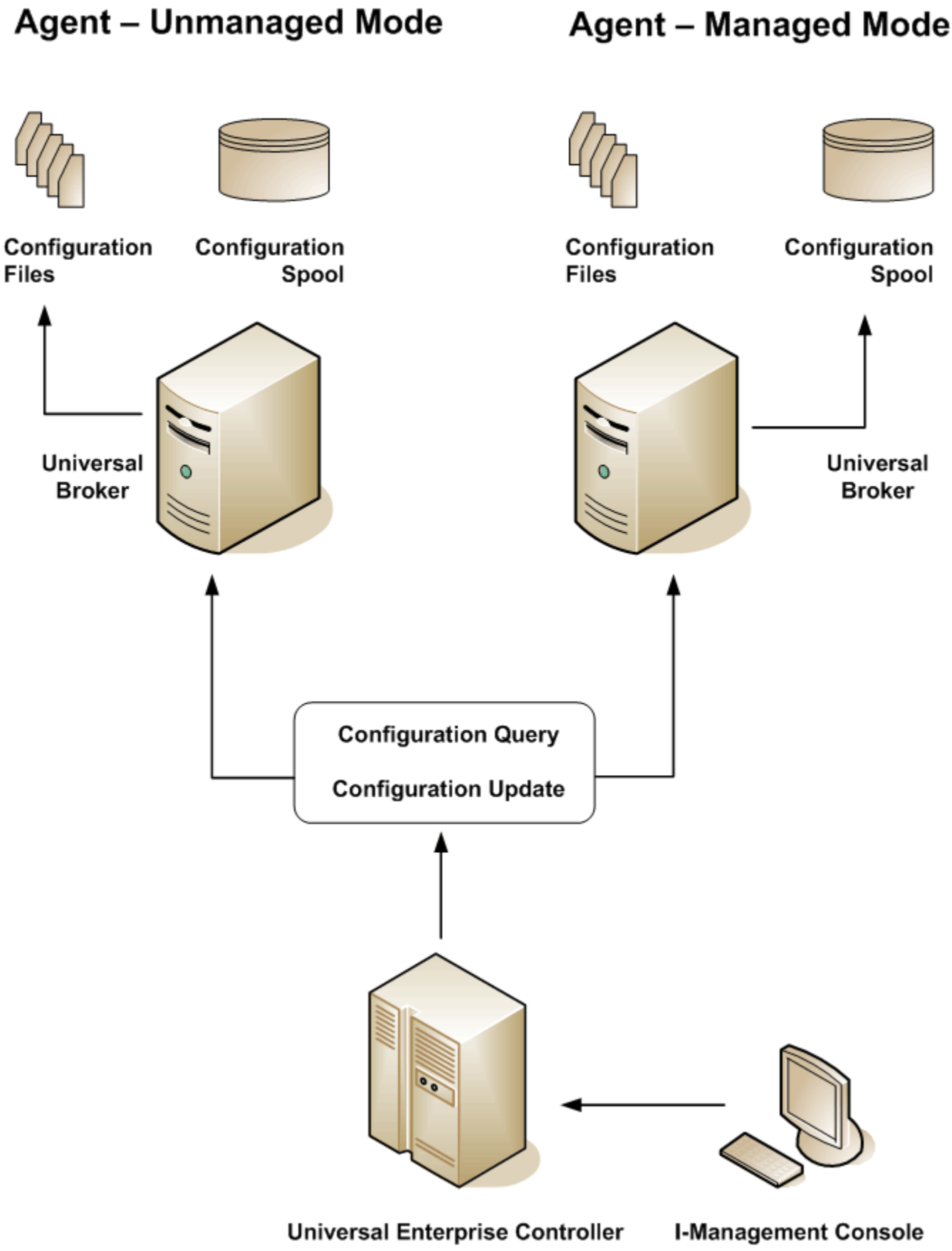
- If a configuration file exists in the configuration directory, it is overwritten.
- If a configuration file does not exist, it is created.

Note
Because of remote configuration and the desire to be able to "lock down" all product configurations, Universal Broker - and all Universal Agent servers - no longer support the command line and environmental variables methods of specifying configuration options.

Selecting Managed Mode

The managed mode of operations for Universal Broker is selected via the I-Administrator client application (see Adding an Agent).

The following figure illustrates remote configuration for one Agent in managed mode and one Agent in unmanaged mode.


Agent – Unmanaged Mode
Agent – Managed Mode
Configuration Files
Configuration Spool
Configuration Files
Configuration Spool
Universal Broker
Universal Broker
Configuration Query
Configuration Update
Universal Enterprise Controller
I-Management Console

Universal Broker Start-up

At Universal Broker start-up, in both managed and unmanaged modes, the Universal Broker configuration file is always read.

Start-up in Unmanaged Mode

At Universal Broker start-up in unmanaged mode, Universal Broker reads the configuration files of all Universal Agent components into its memory. The Universal Broker configuration file is used to define the Universal Broker configuration, just as all configuration files are used in unmanaged mode. Universal Broker updates its memory from the configuration files whenever Universal Control issues a configuration refresh request.

Start-up in Managed Mode

At Universal Broker start-up in managed mode, the Universal Broker configuration file points Universal Broker to the location of the configuration spool file, from which the Broker retrieves configuration information for all Universal Agent components. Universal Broker updates its memory from the configuration spool file and, automatically, after changes are made via I-Management Console.

If more configuration information than needed is included in the Universal Broker configuration file at Universal Broker start-up, Universal Broker will update its running configuration with the information that it retrieved from the spool file. The configuration file that was used at start-up is made obsolete.

Universal Configuration Manager

- Overview
- Availability
- Accessing the Universal Configuration Manager
- Navigating through Universal Configuration Manager
- Modifying / Entering Data
  - Rules for Modifying / Entering Data
- Saving Data
- Installed Components

Overview

The Universal Configuration Manager is a Universal Agent graphical user interface application that enables you to configure all of the Universal Agent for Windows components that have been installed on a Windows operating system.

It is the recommended method of specifying configuration data that will not change with each command invocation. Universal Configuration Manager helps protect the integrity of the configuration file by validating all changes to configuration option values.

Availability

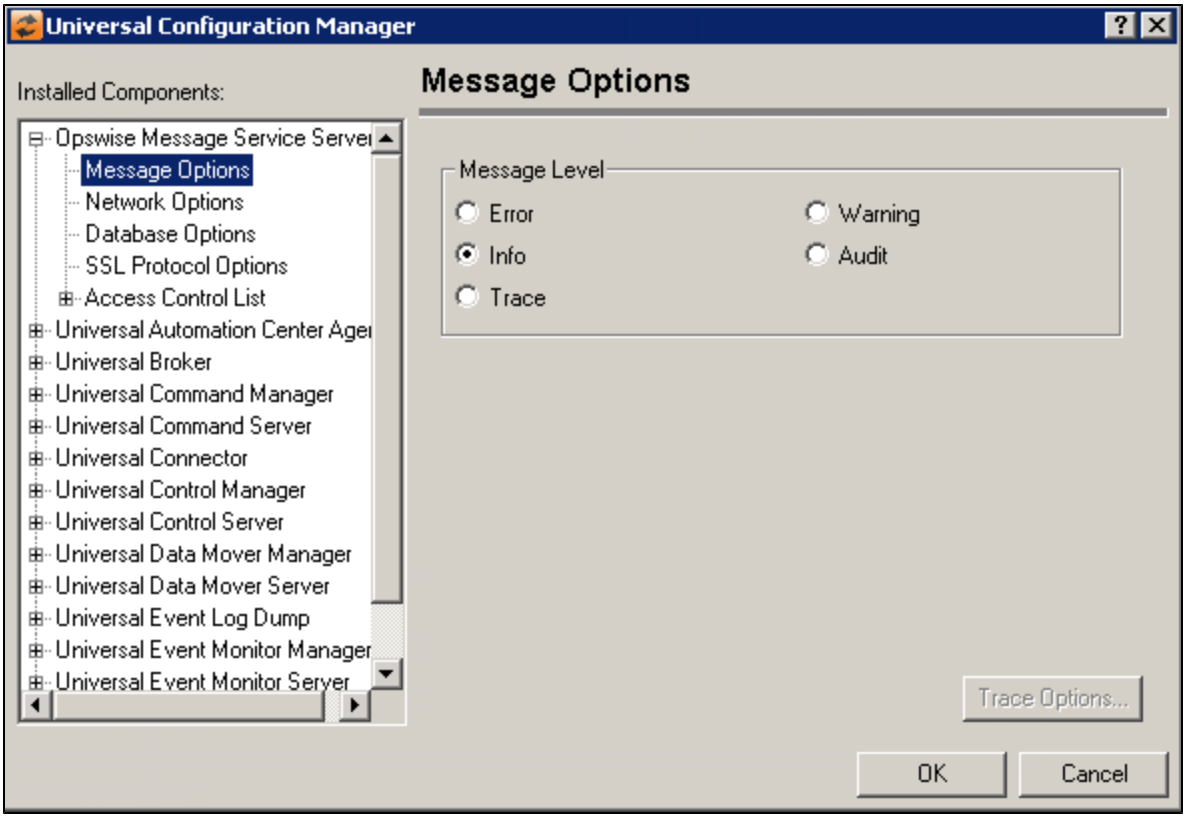
Universal Configuration Manager is installed automatically on the Windows operating system as part of every Universal Agent for Windows installation.

It is available to all user accounts in the Windows Administrator group.

Accessing the Universal Configuration Manager

To access the Universal Configuration Manager:

Step 1	Display the Windows Control Panel.
Step 2	Select the Universal Configuration Manager icon to display the Universal Configuration Manager screen. Note that it may be necessary to switch to the Large or Small icon view in Control Panel to see the icon. It is not likely to show up in the Category view.



Each Universal Configuration Manager screen contains two sections:

1. Left side of the screen displays the Installed Components tree, which lists:
   - Universal Agent components currently installed on your system.
   - Property pages available for each component (as selected), which include one or more of the following:
     - Configuration options
     - Access control lists
     - Licensing information
     - Other component-specific information
2. Right side of the screen displays information for the selected component / page.

(By default, Universal Configuration Manager displays the first property page of the first component in the Installed Components tree.)

Navigating through Universal Configuration Manager

To display general information about a component, click the component name in the Installed Components list.

To display the list of property pages for a component, click the **+** icon next to the component name in the Installed Components list.

To display a property page, click the name of that page in the Installed Components list.

If a property page has one or more of its own pages, a **+** icon displays next to the name of that property page in the Installed Components list. Click that **+** icon to display a list of those pages.

In the previous figure, for example:

- List of property pages is displayed for Universal Broker.
- Message Options property page has been selected, and information for that property is displayed on the right side of the page.
- No **+** icons next to any of the property pages indicates that they do not have one or more of their own property pages.

Modifying / Entering Data

On the property pages, modify / enter data by clicking radio buttons, selecting from drop-down lists, and/or typing in data entry fields.

Some property pages provide panels that you must click in order to:

- Modify or adjust the displayed information.

- Display additional, modifiable information.

> **Note**
> You do not have to click the **OK** button after every modification or entry, or on every property page on which you have modified and/or entered data. Clicking **OK** just once, on any page, will save the modifications and entries made on all pages - and will exit Universal Configuration Manager (see #Saving Data.)

Rules for Modifying / Entering Data

The following rules apply for the modification and entry of data:

- Quotation marks are not required for configuration values that contain spaces.
- Edit controls (used to input free-form text values) handle conversion of any case sensitive configuration values. Except where specifically noted, values entered in all other edit controls are case insensitive.

Saving Data

To save all of the modifications / entries made on all of the property pages, click the **OK** button at the bottom of any property page. The information is saved in the configuration file, and Universal Broker is automatically refreshed.

Clicking the **OK** button also exits the Universal Configuration Manager. (If you click **OK** after every modification, you will have to re-access Universal Configuration Manager.)

To exit Universal Configuration Manager without saving any of the modifications / entries made on all property pages, click the **Cancel** button.

Installed Components

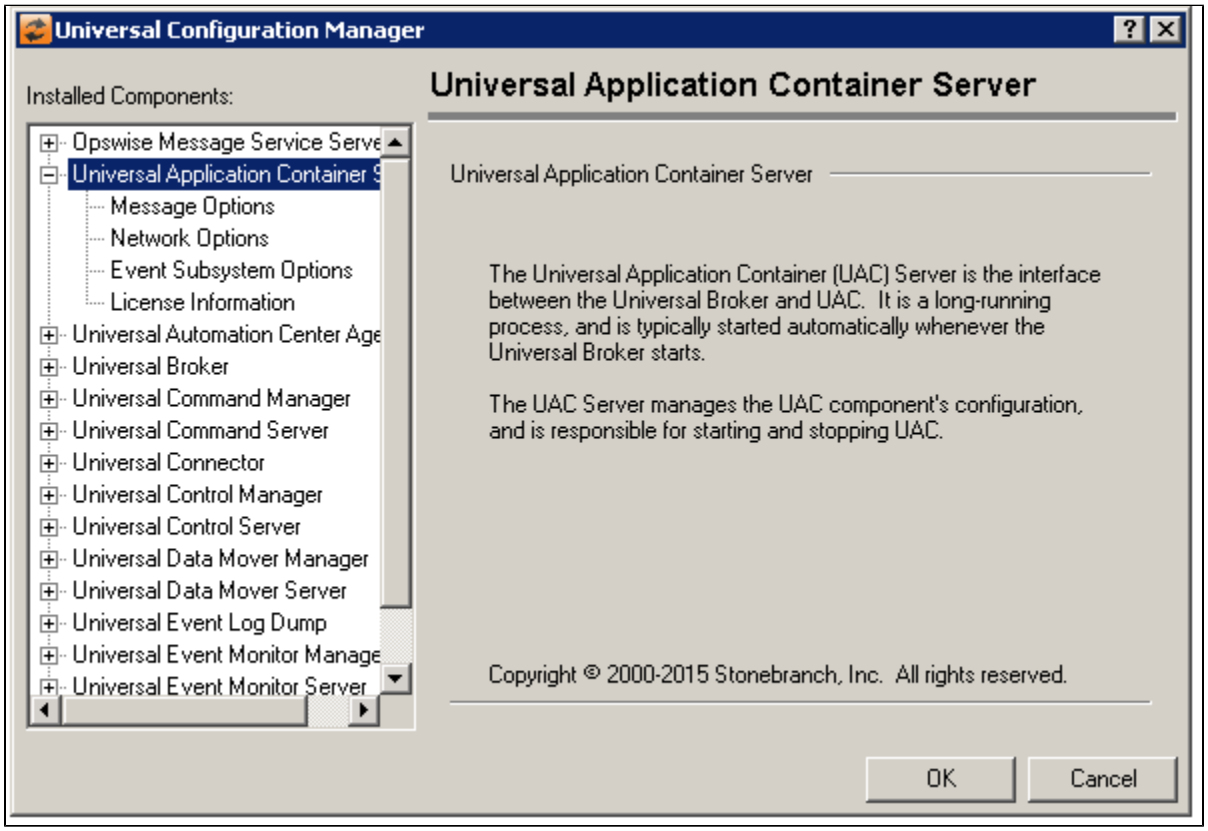
The following pages provide detailed information for all installed components that are configurable via the Universal Configuration Manager:

- Universal Configuration Manager - Universal Application Container Server
- Universal Configuration Manager - Universal Automation Center Agent (UAG)
- Universal Configuration Manager - Universal Broker
- Universal Configuration Manager - Universal Command Manager
- Universal Configuration Manager - Universal Command Server
- Universal Configuration Manager - Universal Connector for SAP
- Universal Configuration Manager - Universal Control Manager
- Universal Configuration Manager - Universal Control Server
- Universal Configuration Manager - Universal Data Mover Manager
- Universal Configuration Manager - Universal Data Mover Server
- Universal Configuration Manager - Universal Event Log Dump
- Universal Configuration Manager - Universal Event Monitor Manager
- Universal Configuration Manager - Universal Event Monitor Server
- Universal Configuration Manager - Universal Message Service (OMS)
- Universal Configuration Manager - Universal Query

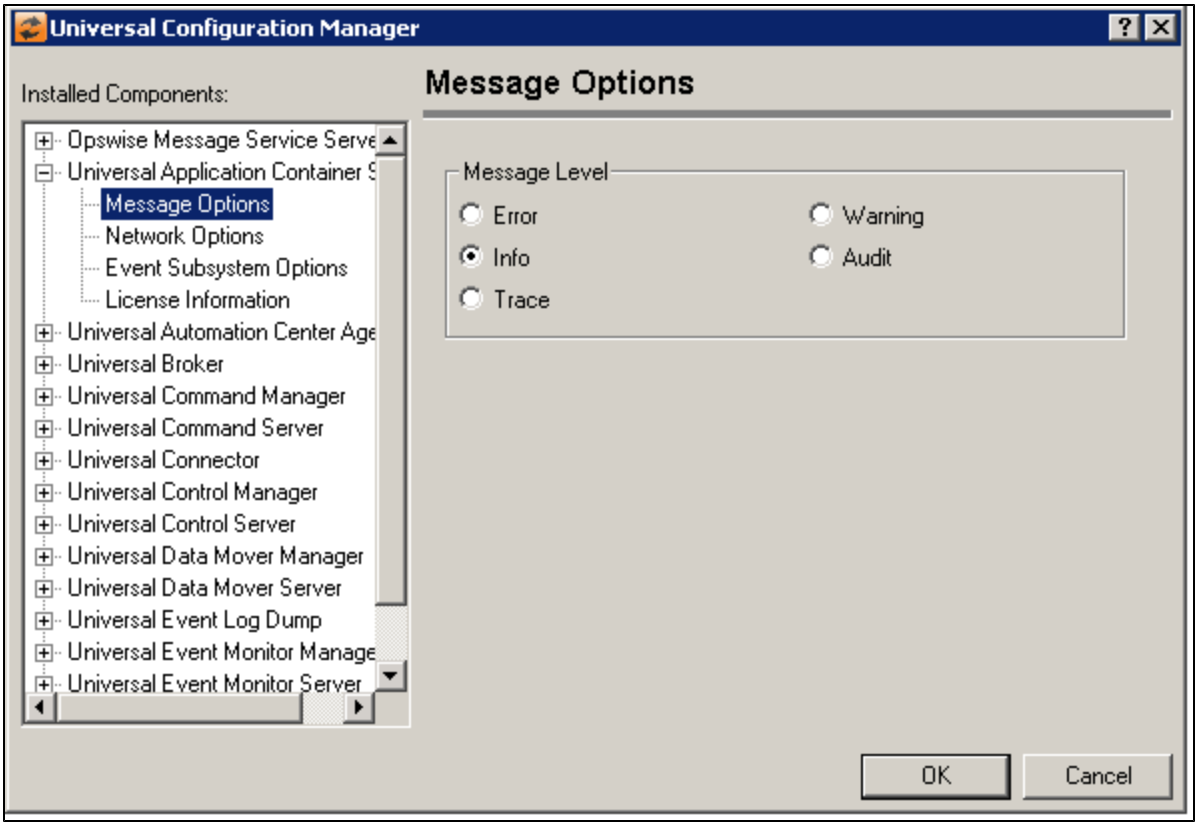
Universal Configuration Manager - Universal Application Container Server

- Universal Application Container Server
- Universal Application Container Server: Message Options
- Universal Application Container Server: Network Options
- Universal Application Container Server: Event Subsystem Options
- Universal Application Container Server: License Information

Universal Application Container Server

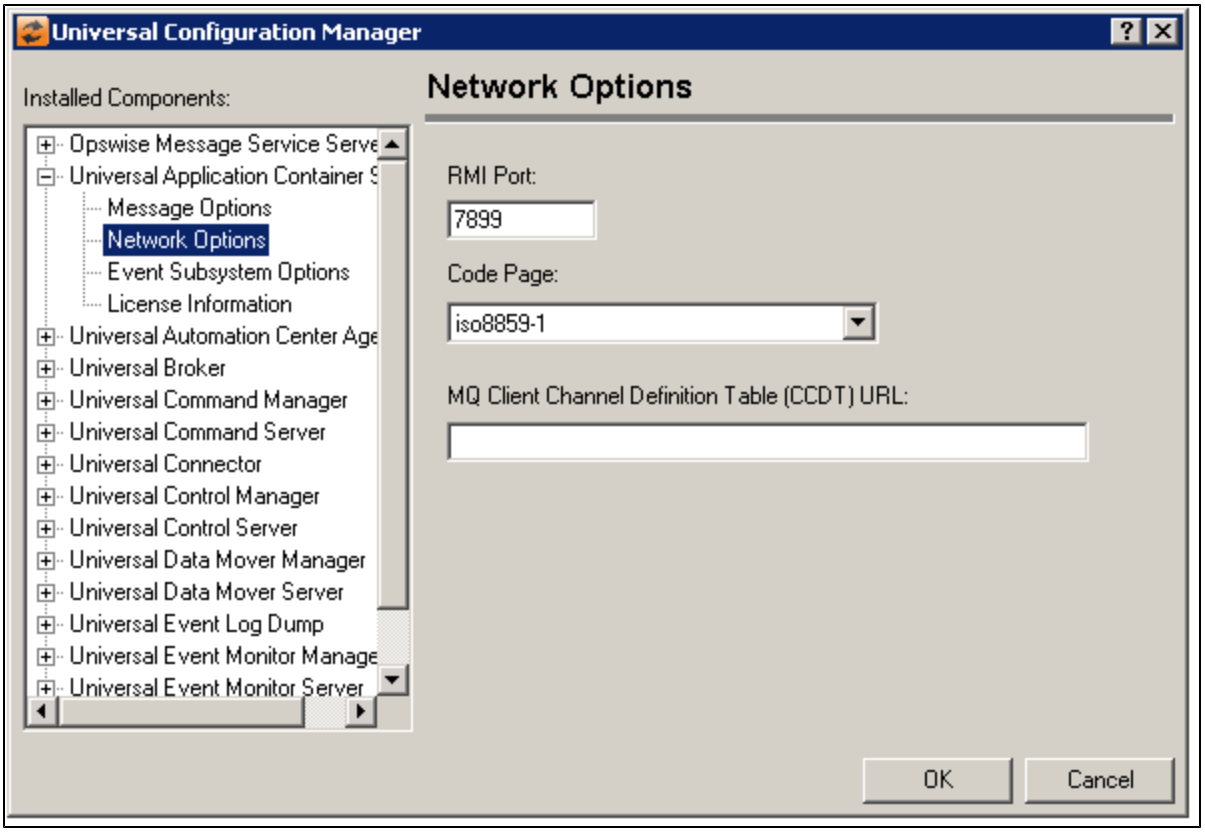


Universal Application Container Server: Message Options



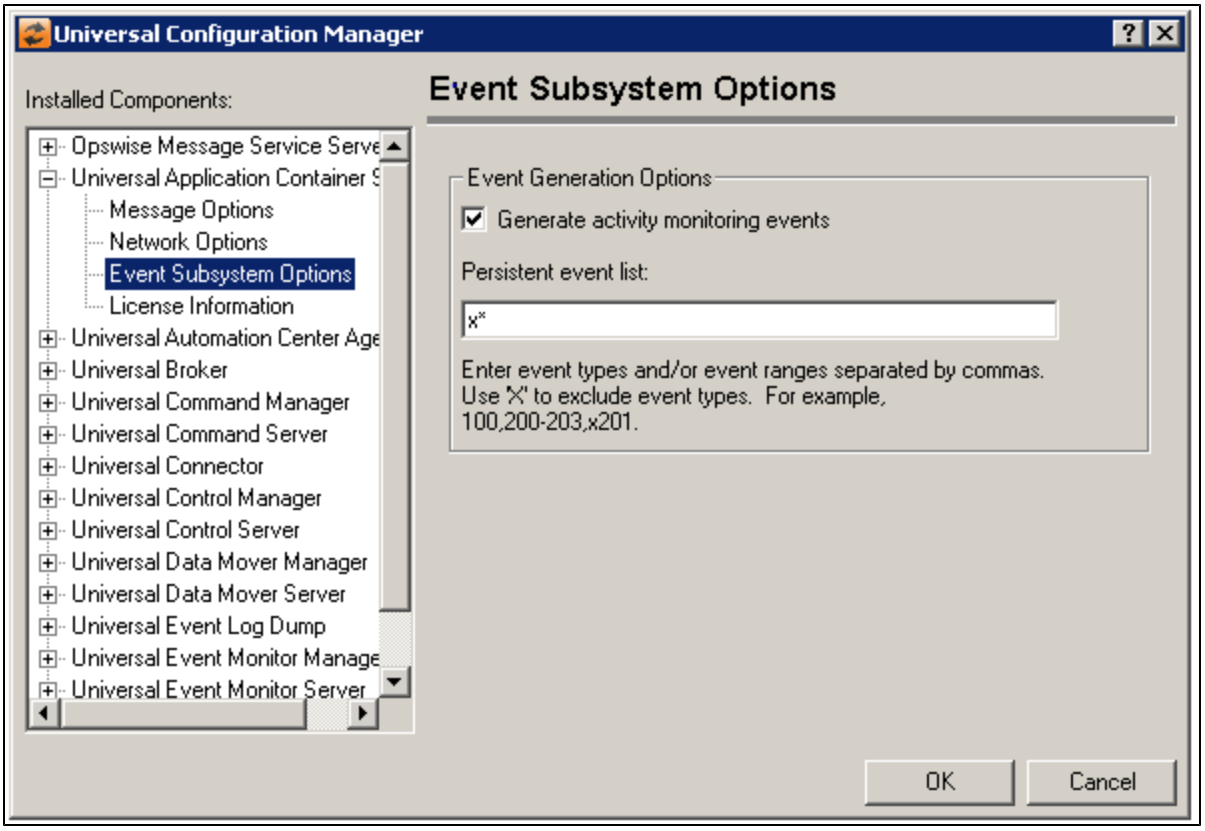
Field	Description
Message Level	**Error** Turns on message logging for errors only. **Warning** Turns on message logging for warnings and errors. **Info** Turns on message logging for all informational, warning and error messages. **Audit** Turns on message logging for all audit, informational, warning and error messages. **Trace** Allows all messages to be logged, including those used for debugging purposes.

Universal Application Container Server: Network Options



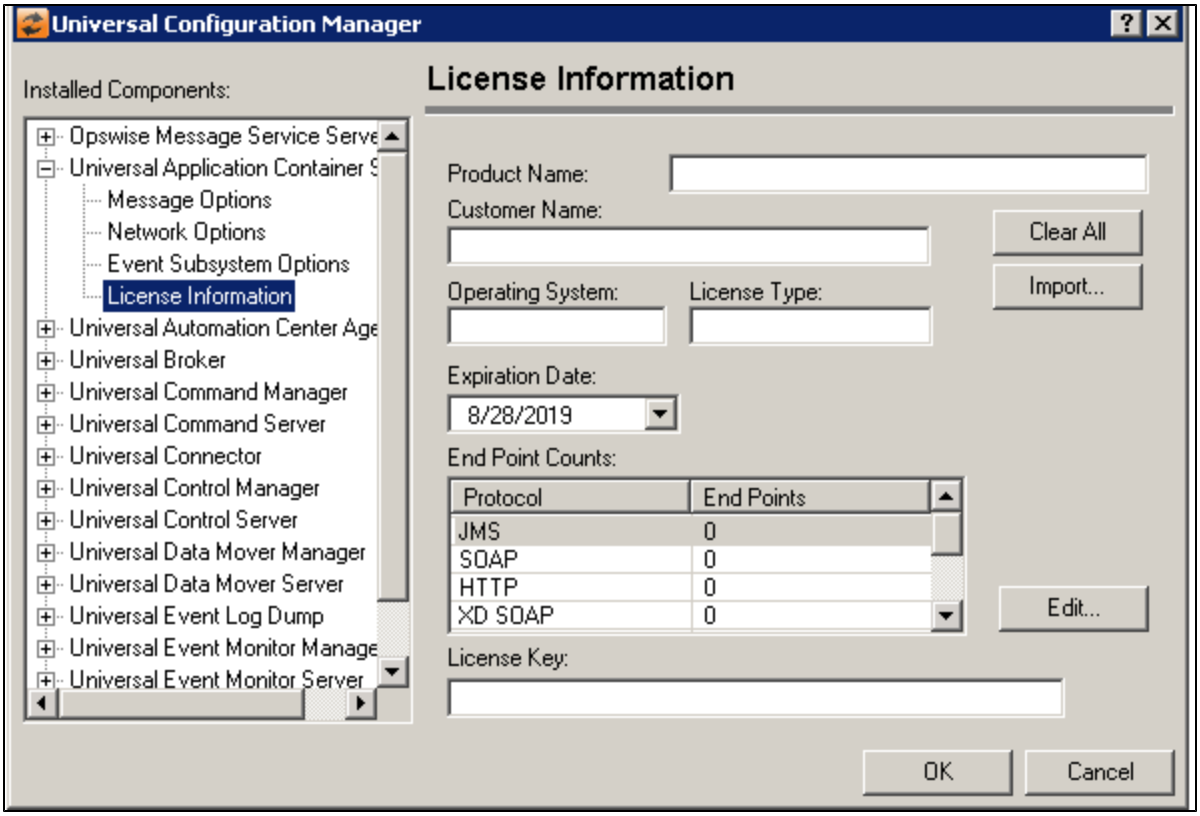
Field	Description
RMI Port	The Remote Method Invocation (RMI) port number specifies the port number or service name on which UAC will listen for service requests from UAC Server.
Code Page	Sets the translation table used to translate text-based data transmitted across the network.
MQ Client Channel Definition Table (CCDT) URL:	Specifies the location of an optional MQ Client Channel Definition Table (CCDT) that can be used to establish client connections to remote MQ Managers. The value is a URL that specifies the location of the CCDT file. The URL may point to: • Local file system; for example: file://pathname/file • FTP server; for example: ftp://host/pathname/file

Universal Application Container Server: Event Subsystem Options



Field	Description
Event Generation Options	**Generate activity monitoring events** When this option is selected, the Universal Application Container Server will generate events that monitor product activity. This option is selected by default. **Persistent event list** Specifies which events are to be generated and processed as persistent events. A persistent event is saved in a Universal Enterprise Controller (UEC) event database for long-term storage. Enter a list and/or range of numeric event types, separated by a comma. To exclude an event or range of events, precede the entry with an upper- or lower-case 'X'. Use an asterisk (*) to specify all event types. For example: • x* Do not generate any events (this is the default) • * Generate all event types • 100 Generate all event types except 100 • 100,200-205 Generate event type 100 and 200 through 205 • 100,200-205,x202 Generate event types 100 and 200 thru 205, excluding event type 202 Entries are processed from left to right, and the list is order dependent. For example, an entry of x*,100 prevents generation of all event types except 100, while an entry of 100,x* prevents generation of all event types (the right-most entry takes precedence).

Universal Application Container Server: License Information

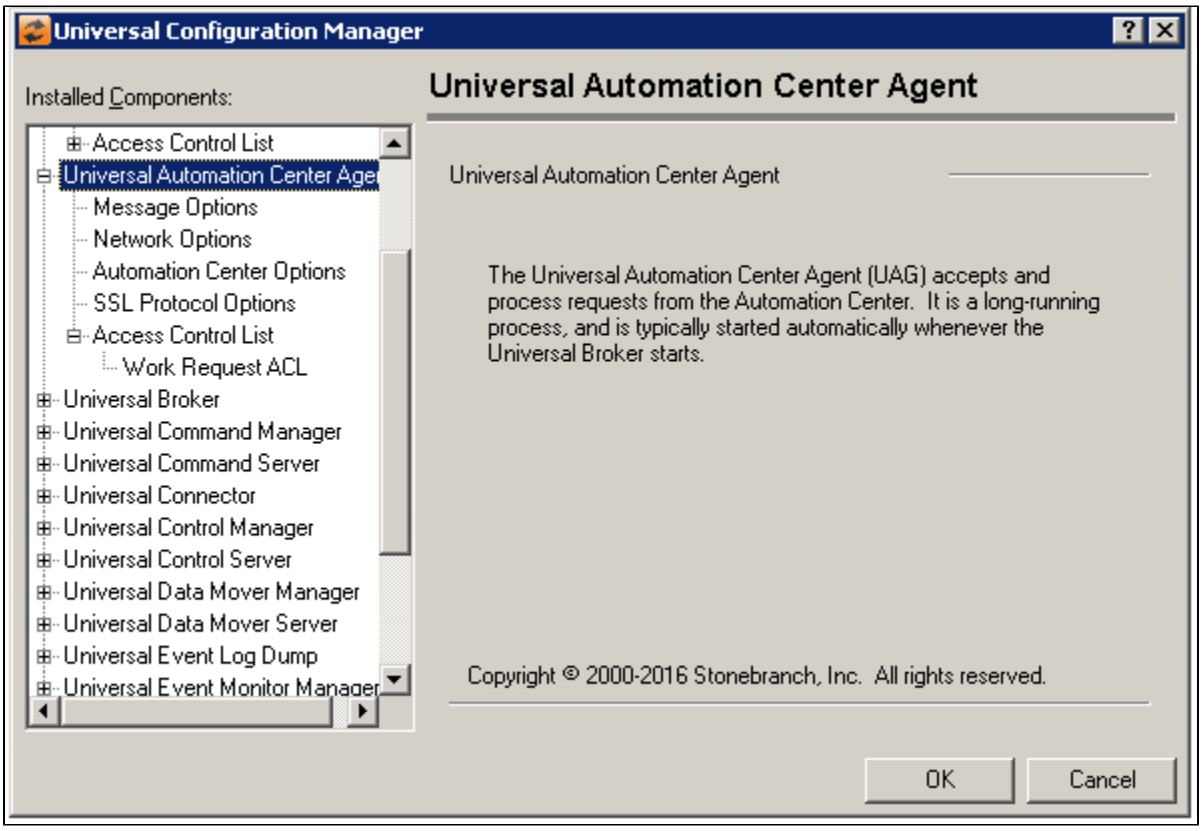


Field	Description
Product Name	Specifies the name of the registered product for which the license key was issued.
Customer Name	Contains the name under which the software is registered. This value must be entered exactly as the customer name specified in the licensing material you received with the software.
Operating System	Identifies the operating system on which your licensed Universal Application Container Server will run. This value should match the operating system specified in the licensing material you received with your software.
License Type	Specifies the type of license issued for the current product. This value is used with the expiration date to determine the length of time the software license is valid.
Expiration Date	Contains the date when the current license for this product will expire. The date entered must match the expiration date specified in the licensing material you received with the software. To change the date, enter a new value, or click the down arrow to display a calendar control.
End Point Counts	Lists supported protocols and the number of licensed end points for each. The number of end points for each protocol should match the numbers specified in the licensing material you received with your software.
License Key	Contains the license key that is generated for each product, customer, license type and expiration date. The value contained in this field must match the key specified in the licensing material you received with the software package.
Clear All Button	Select this button to reset the license information fields to their initialized defaults.
Edit Button	Select this button to display a dialog from which the number of licensed end points for the selected protocol can be updated.
Import Button	Select this button to import a text file that contains formatted license information.

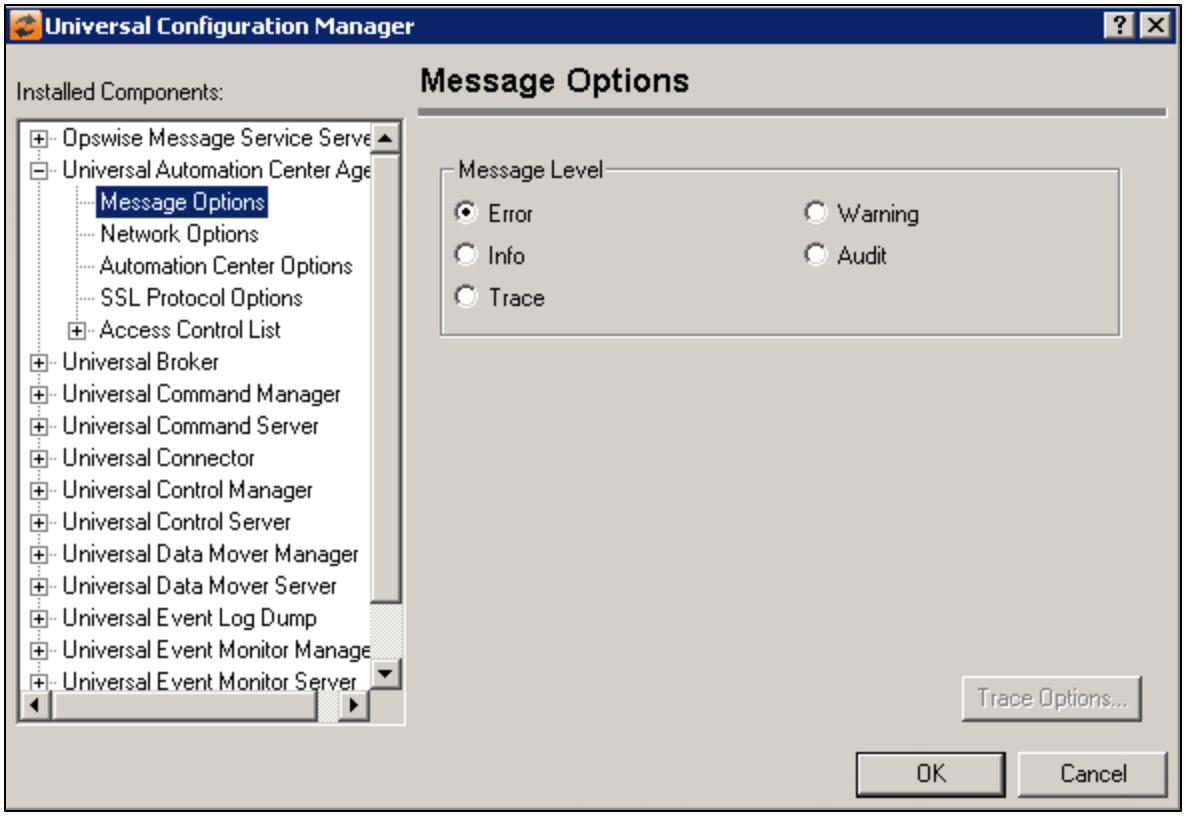
Universal Configuration Manager - Universal Automation Center Agent (UAG)

- Universal Automation Center Agent (UAG)
- Universal Automation Center Agent: Message Options
- Universal Automation Center Agent: Network Options
- Universal Automation Center Agent: Automation Center Options
- Universal Automation Center Agent: SSL/TLS Protocol Options
- Universal Automation Center Agent: Access Control List
  - Universal Automation Center Agent: Access Control List: Work Request ACL

Universal Automation Center Agent (UAG)

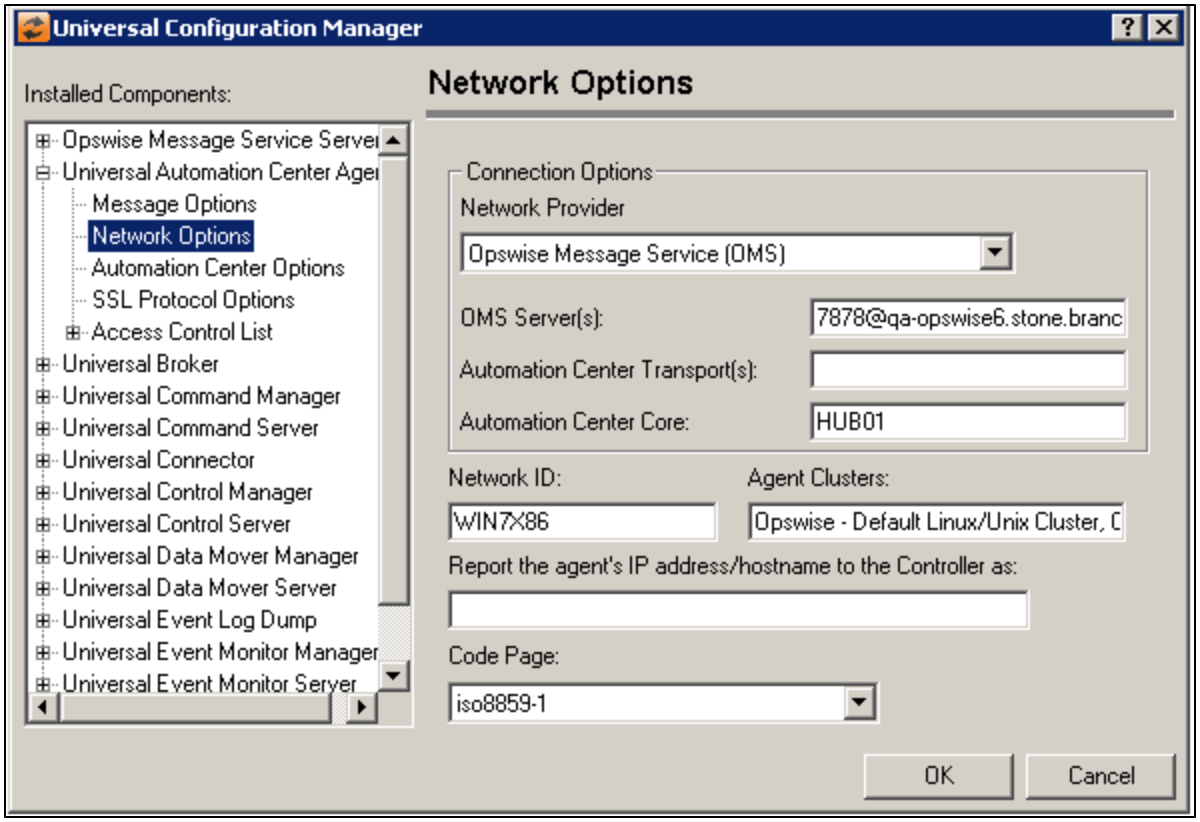


Universal Automation Center Agent: Message Options



Field	Description
Message Level	**Error** Turns on message logging for errors only. **Warning** Turns on message logging for warnings and errors. **Info** Turns on message logging for all informational, warning and error messages. **Audit** Turns on message logging for all audit, informational, warning and error messages. **Trace** Allows all messages to be logged, including those used for debugging purposes.
Trace Options Button	Displays a dialog that allows options for trace statement buffering and trace file limits to be set.

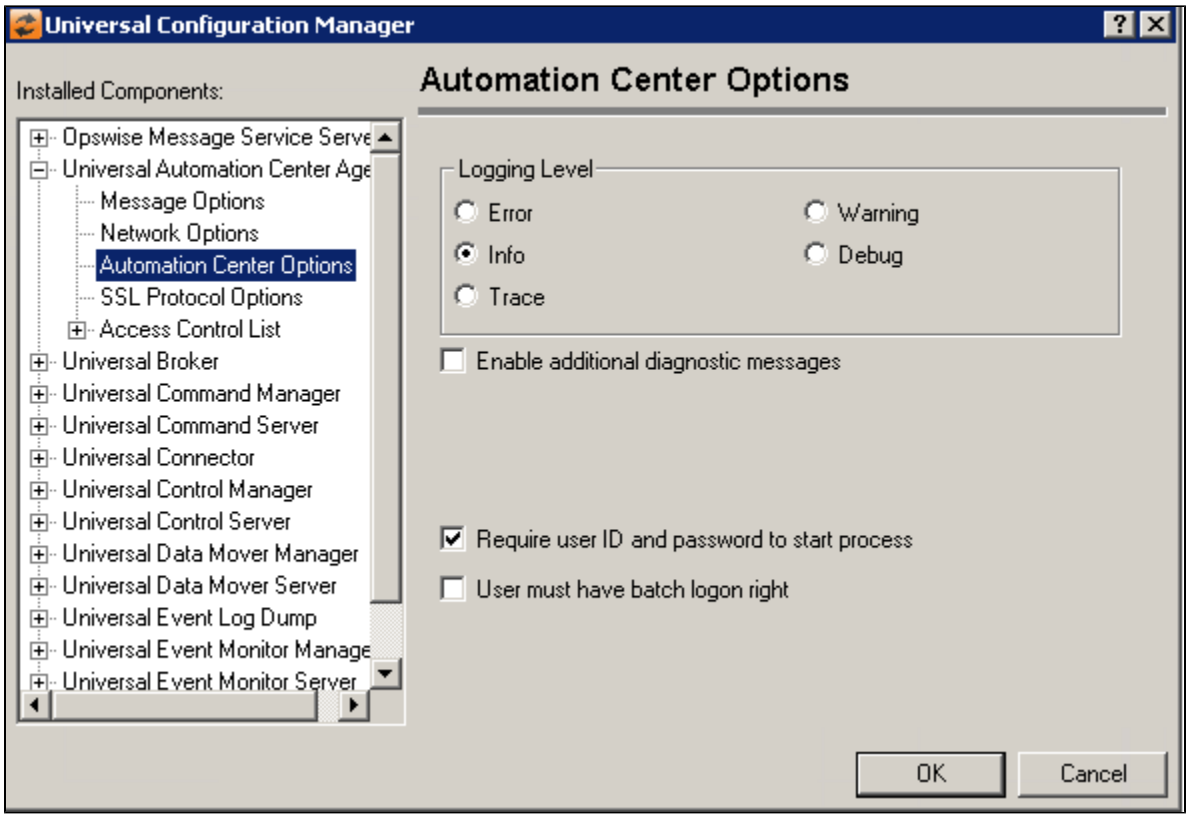
Universal Automation Center Agent: Network Options



Field	Description
Connection Options	**Network Provider** Specifies the method UAG uses to communicate with the Opswise Controller. Select Opswise Message Service to connect via an OMS Server. This is the preferred communication method. Select Opswise Transport and Message Hub to connect to the Controller using values for the Automation Center Transport and Automation Center Core. This method will be deprecated in a future release. If no network provider is selected, the Automation Center Agent determines the method based on the connection options that are provided. **OMS Server(s)** Specifies the port and network address of an Opswise Message Service (OMS) Server that the agent will use for communication with an Opswise Controller. This value is required if the Opswise Message Service network provider is chosen in the list above. The syntax of the value is port@host[,port@host...], where: • port is the TCP port number on which the OMS Server is listening. • host is the host name or IP address of the OMS Server. If multiple servers are specified, they are separated by a comma. **Automation Center Transport(s)** Automation Center transports specifies the port and network address of Automation Center transports used for network communication. This is a required field if a network provider of Opswise Transport and Message Hub is selected. The syntax of the value is port@host[,port@host...], where: • port is the TCP port number on which the transport is listening. • host is the host name or IP address of the transport. If multiple transports are specified, they are separated by a comma. **Automation Center Core** Automation Center core specifies the queue name for the Automation Center message hub. This is a required field if a network provider of Opswise Transport and Message Hub is selected.

Network ID	Set the default network ID of the agent. When set to the default of AUTOCONF, the agent requests an ID from the automation center core following the first successful connection attempt.
Agent Clusters	Specifies a comma-separated list of clusters that the Automation Center Agent will join.
Report the agents IP Address / host name to the controller as	The hostname or IP address that the Agent should report as its outbound network interface to the Controller.
Code Page	Sets the translation table used to translate text-based data transmitted across the network.

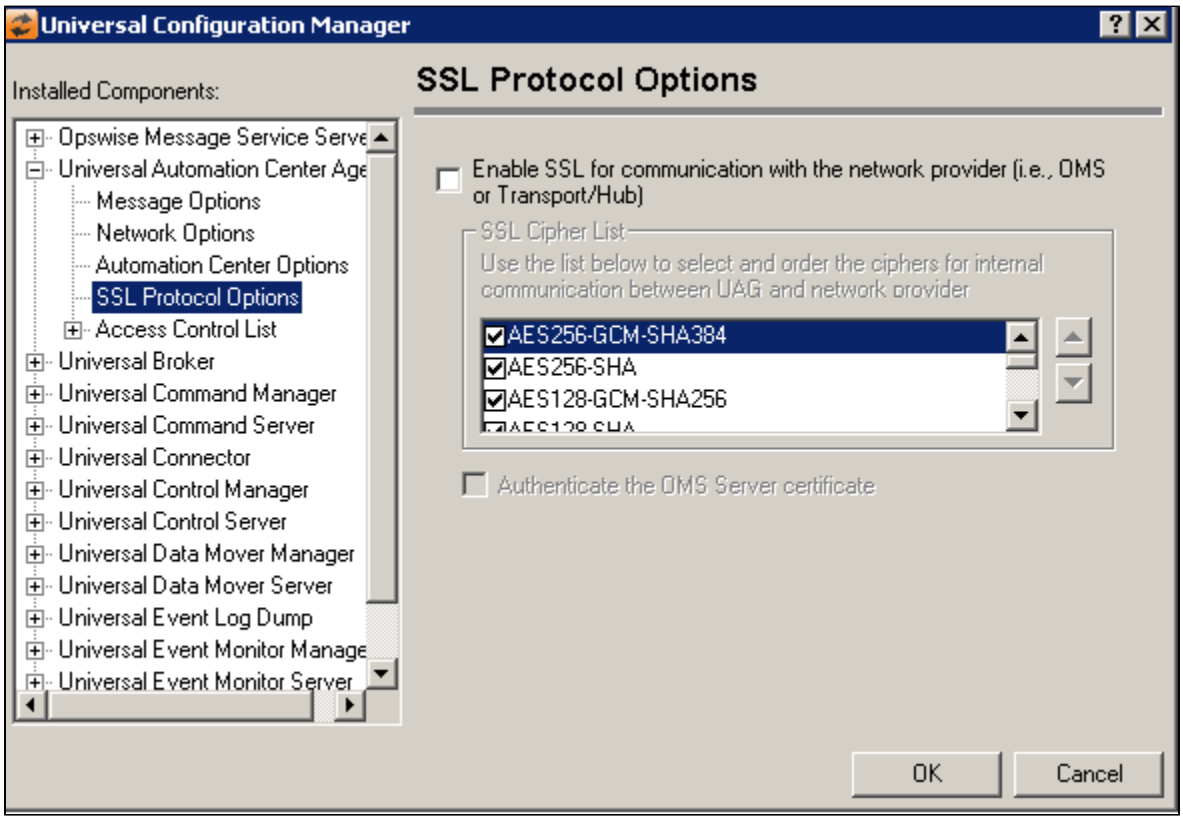
Universal Automation Center Agent: Automation Center Options



Field	Description
Logging Level	**Error** Turns on agent logging for errors only. **Warning** Turns on agent logging for warnings and errors. **Info** Turns on agent logging for all informational, warning and error messages. **Debug** Turns on agent logging for all debug, informational, warning and error messages. **Trace** Allows all agent messages to be logged, including those used for debugging purposes.
Enable additional diagnostic messages	When checked, this option enables additional diagnostic messages and sets the txtdebug configuration option to Y. When this option is not checked, the Configuration Manager sets the txtdebug configuration option to N and disables additional diagnostic Agent messages.
	When checked, this option sets the security configuration option to default to activate user security. When this option is not checked, the security configuration option is set to none. When user security is activated, the Agent requires a local user ID and password so that it may execute commands in the specified user's security context.

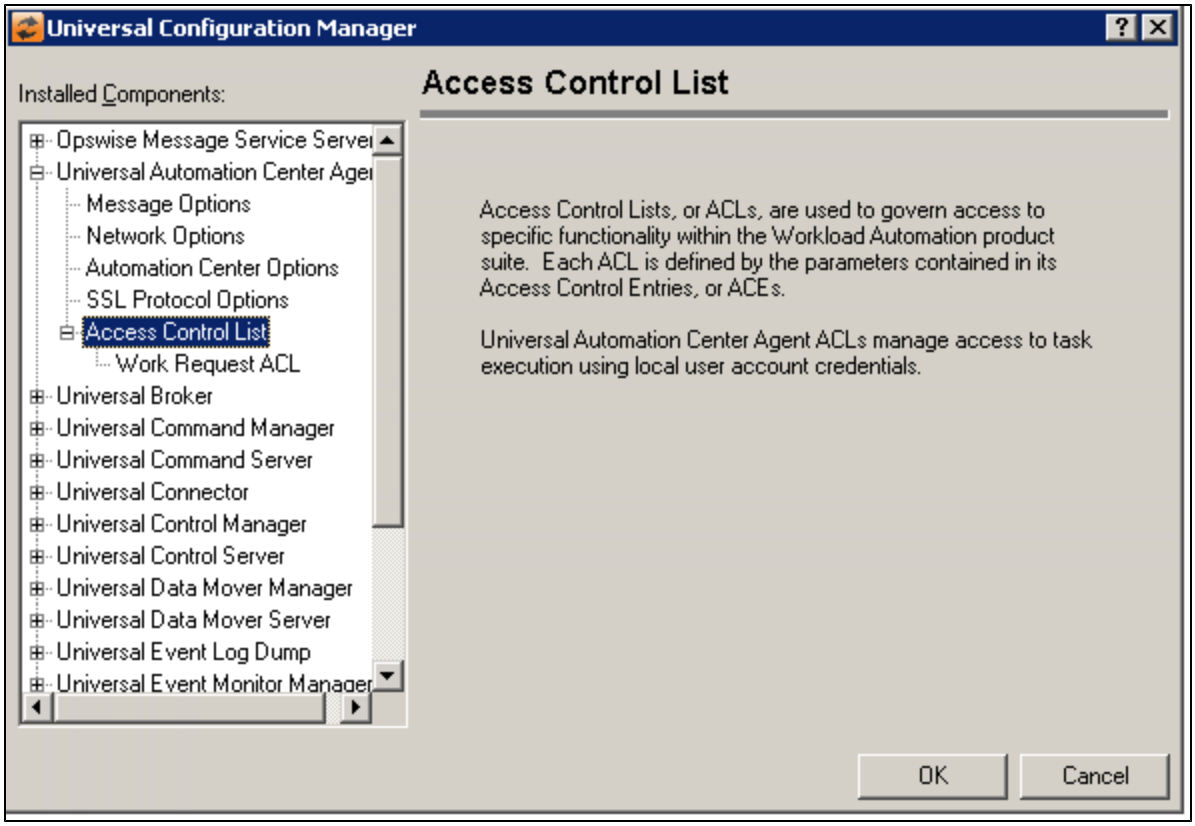
Require user ID and password to start	
User must have batch logon right	When checked, the Log on as a batch job user right must be granted to the user ID specified in the task's credentials before the process will run.

Universal Automation Center Agent: SSL/TLS Protocol Options

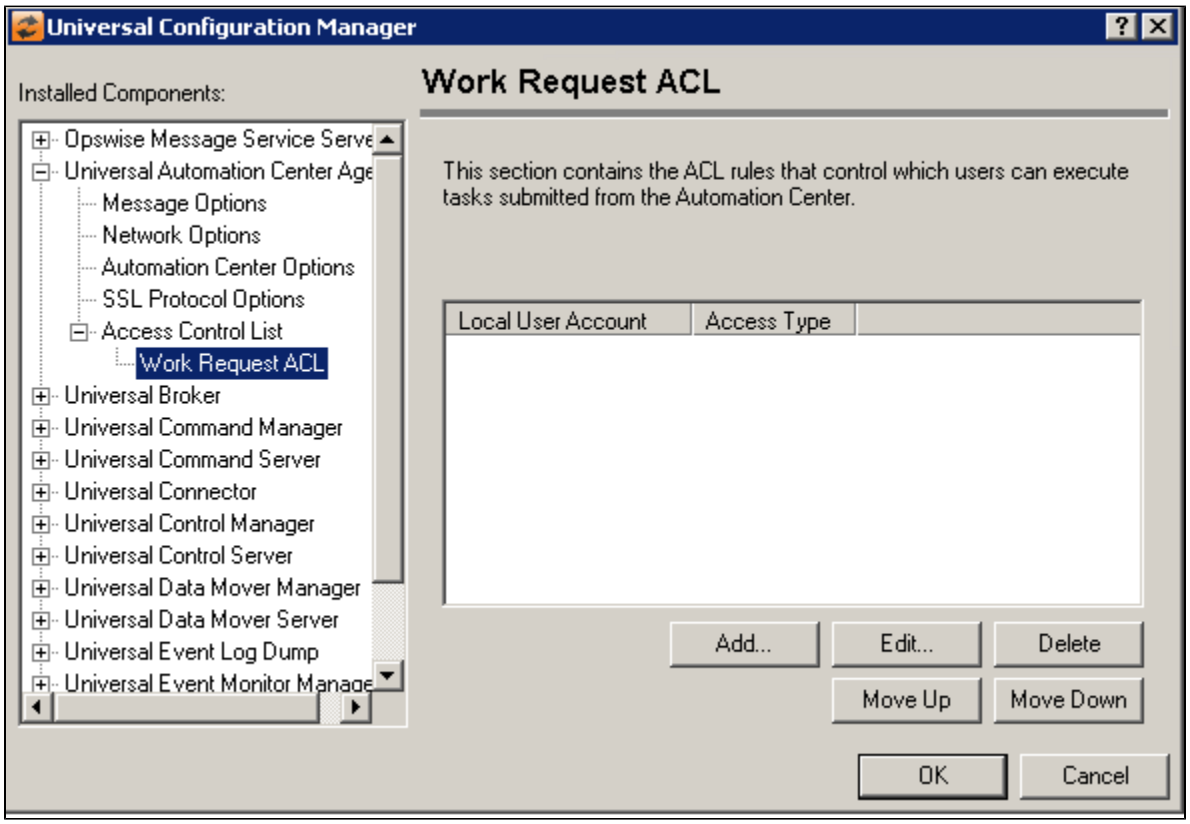


Field	Description
Enable SSL for communication with the network provider	When checked, this option will allow messages exchanged with the network provider (i.e., an OMS Server or Transport /Hub) to be encrypted using SSL/TLS. In order for messages to be encrypted, the network provider must also be configured to use SSL/TLS. If the network provider is not configured to use SSL/TLS, checking this option will have no effect. The default for this option is unchecked.
SSL Cipher List	The SSL/TLS cipher list specifies one or more acceptable cipher suites to use for network communication. The list consists of one or more cipher suites ordered with the most preferred suite first and the least preferred suite last.
Authenticate the OMS Server certificate	When checked, this option causes the UAG Server to authenticate an OMS Server's certificate during SSL/TLS session creation. The OMS Server must be configured to use certificates before messages are encrypted. If the network provider is not OMS or the OMS Server is not configured to use certificates, checking this option will have no effect. The default for this option is unchecked.

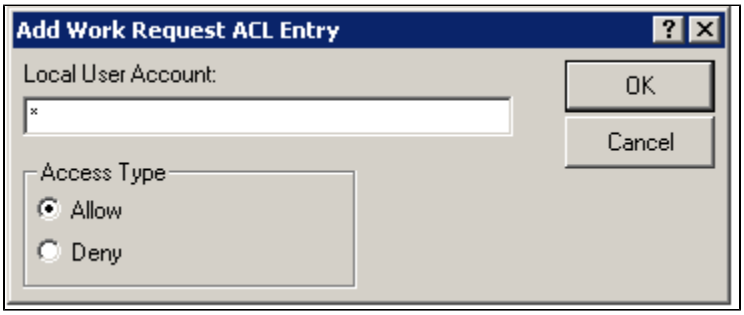
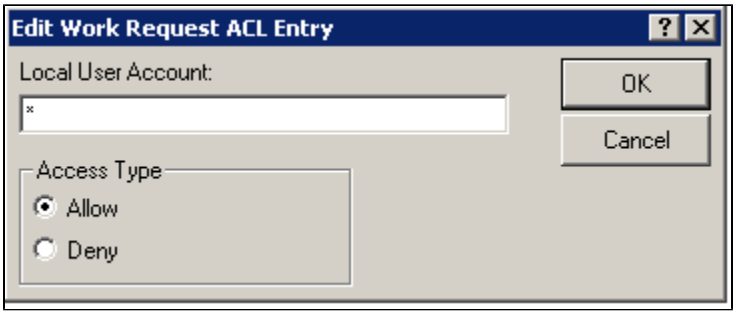
Universal Automation Center Agent: Access Control List



Universal Automation Center Agent: Access Control List: Work Request ACL



Field	Description
Work Request ACL List	List of ACL rules that control which users can execute tasks submitted from the Automation Center.

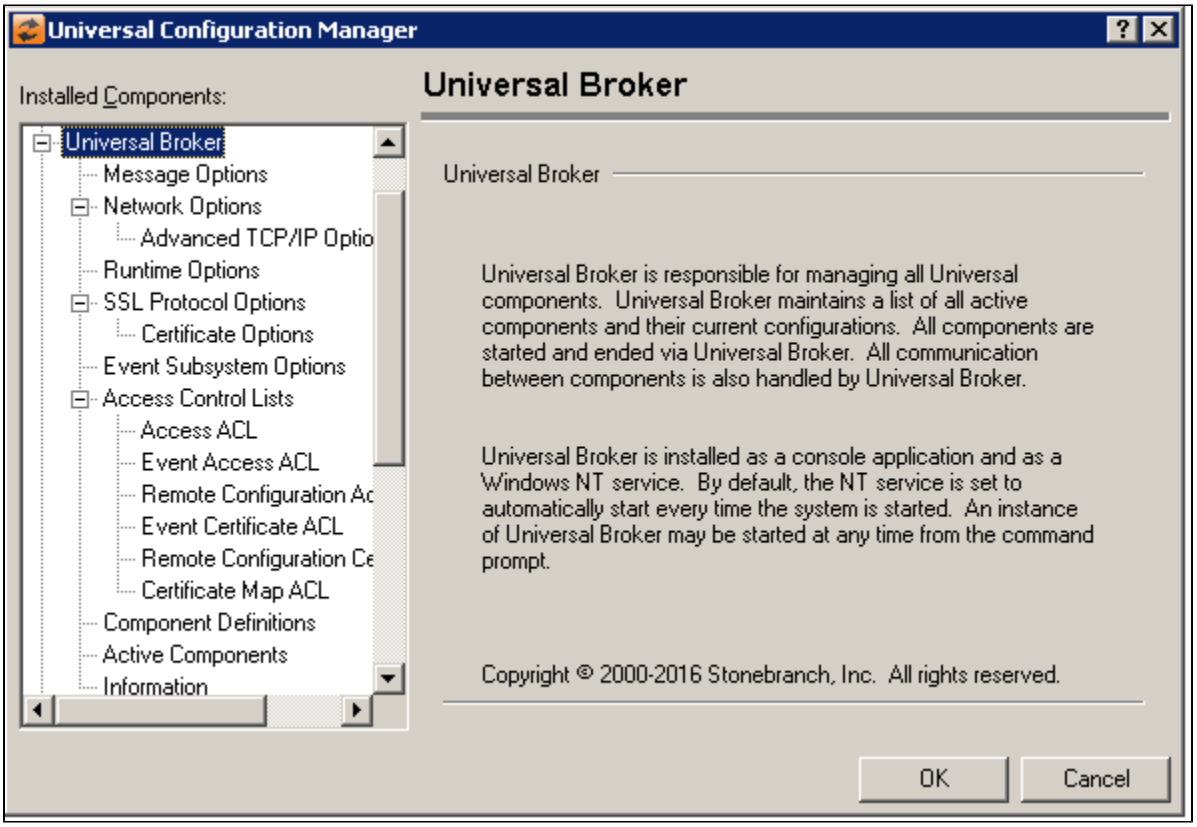
Add Button	Displays a dialog box that allows an entry to be added to the Access Control List. The fields in this dialog are populated with default values.  **Local User Account** The local user account that will be used to run the requested process. **Access Type** Grants/Prevents the specified local user permission to execute tasks on this system that were submitted via the Workload Automation Center.
Edit Button	Displays a dialog that allows the selected Access Control List entry to be modified.  **Local User Account** The local user account that will be used to run the requested process. **Access Type** Grants/Prevents the specified local user permission to execute tasks on this system that were submitted via the Workload Automation Center.
Delete Button	Removes the selected item from the Access Control List.
Move Up Button	Moves the selected item up one position in the list.
Move Down Button	Moves the selected item down one position in the list.

Universal Configuration Manager - Universal Broker

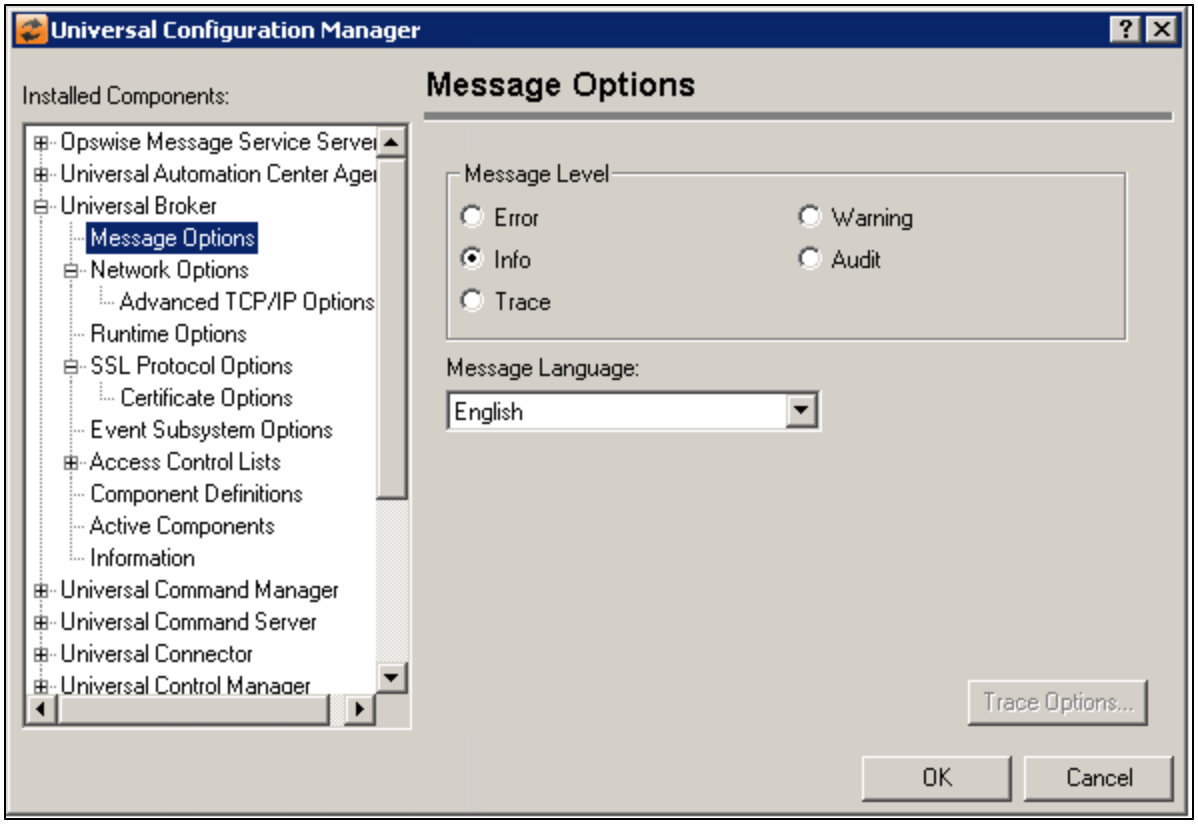
- Universal Broker
- Universal Broker: Message Options
- Universal Broker: Network Options
  - Universal Broker: Network Options - Advanced TCP/IP Options
- Universal Broker: Runtime Options
- Universal Broker: SSL/TLS Protocol Options
  - Universal Broker: SSL/TLS Protocol Options - Certificate Options
- Universal Broker: Event Subsystem Options
- Universal Broker: Access Control Lists
  - Universal Broker: Access Control Lists - Access ACL
  - Universal Broker: Access Control Lists - Event Access ACL
  - Universal Broker: Access Control Lists - Remote Configuration Access ACL
  - Universal Broker: Access Control Lists - Event Certificate ACL
  - Universal Broker: Access Control Lists - Remote Configuration Certificate ACL
  - Universal Broker: Access Control Lists - Certificate Map ACL
- Universal Broker: Component Definitions

- Universal Broker: Active Components
- Universal Broker: Information

Universal Broker

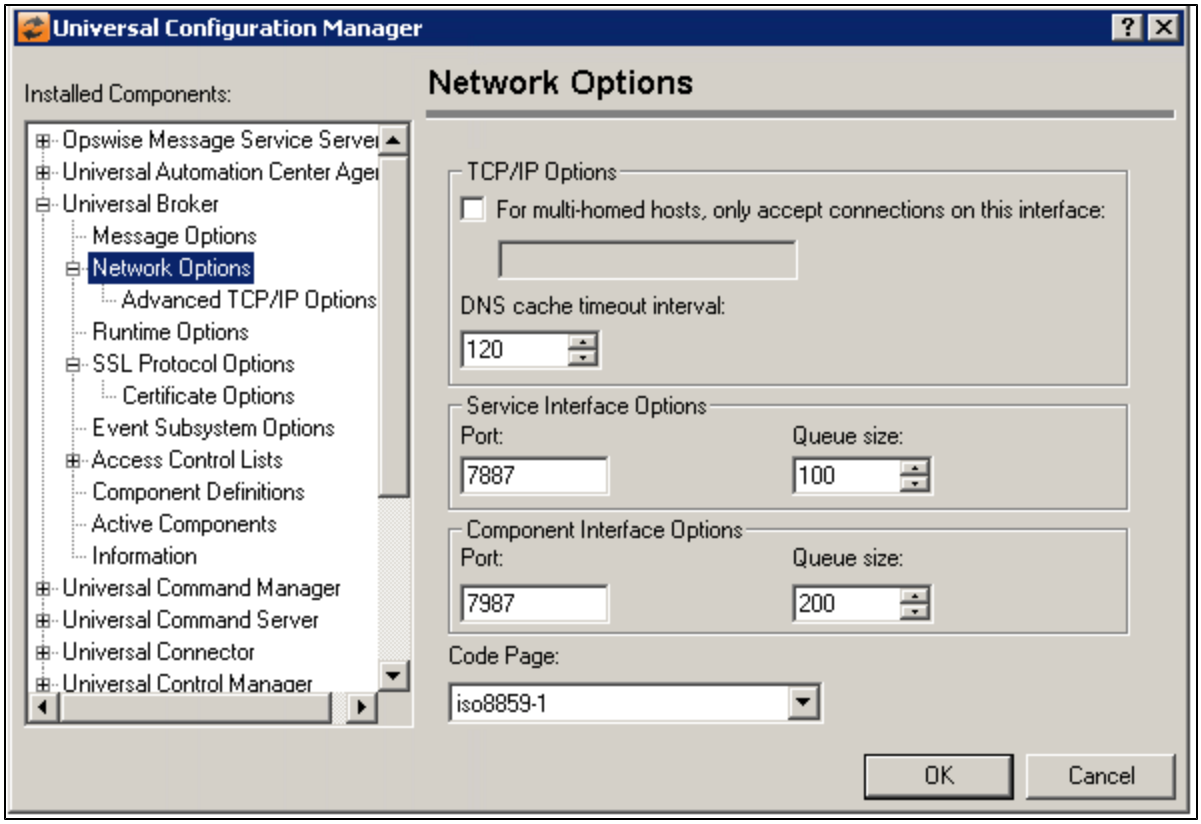


Universal Broker: Message Options

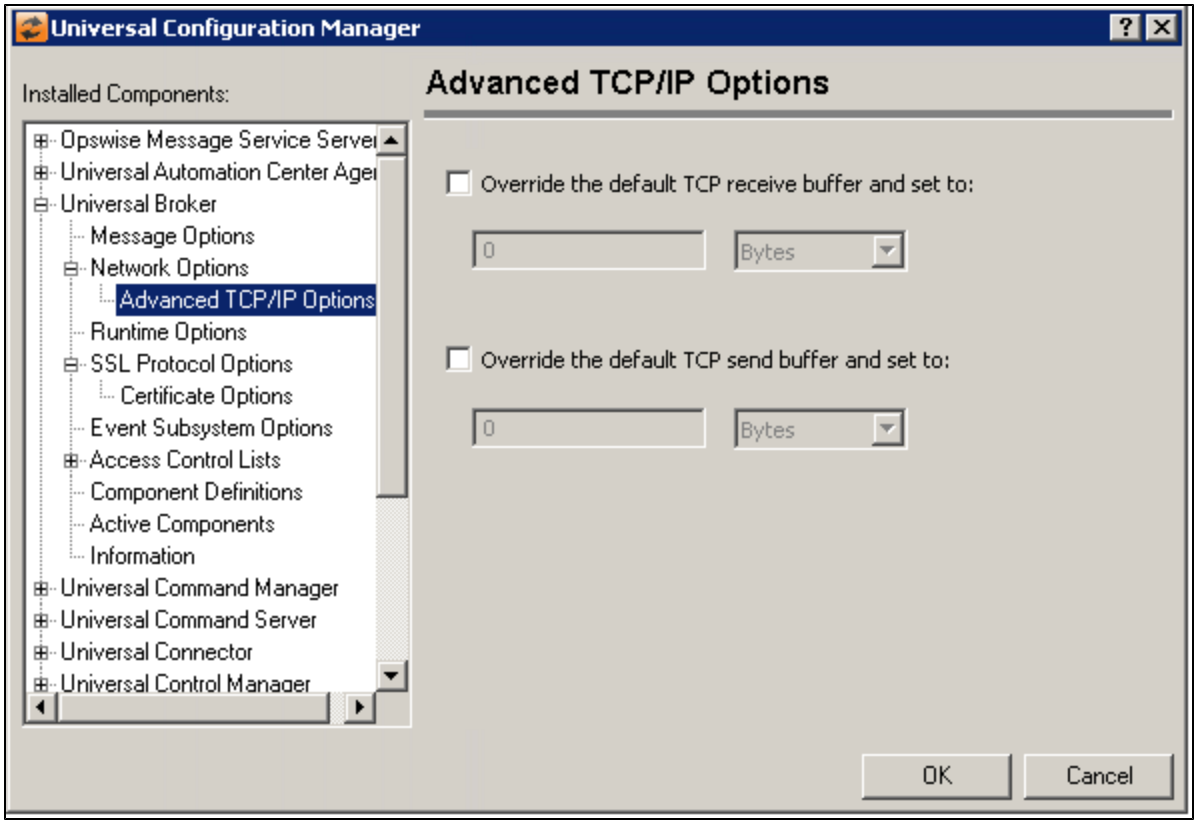


Field	Description
Message Level	**Error** Turns on message logging for errors only. Changes made here to the message level only affect the Broker when it is run as an NT Service. When the Broker is run as a console application, a command line parameter must be used to change the default message level. **Warning** Turns on message logging for warnings and errors. Changes made here to the message level only affect the Broker when it is run as an NT Service. When the Broker is run as a console application, a command line parameter must be used to change the default message level. **Info** Turns on message logging for all informational, warning and error messages. Changes made here to the message level only affect the Broker when it is run as an NT Service. When the Broker is run as a console application, a command line parameter must be used to change the default message level. **Audit** Turns on message logging for all audit, informational, warning and error messages. Changes made here to the message level only affect the Broker when it is run as an NT Service. When the Broker is run as a console application, a command line parameter must be used to change the default message level. **Trace** Allows all messages to be logged, including those used for debugging purposes. Changes made here to the message level only affect the Broker when it is run as an NT Service. When the Broker is run as a console application, a command line parameter must be used to change the default message level.
Message Language	Specifies the language used to display error messages.

Universal Broker: Network Options

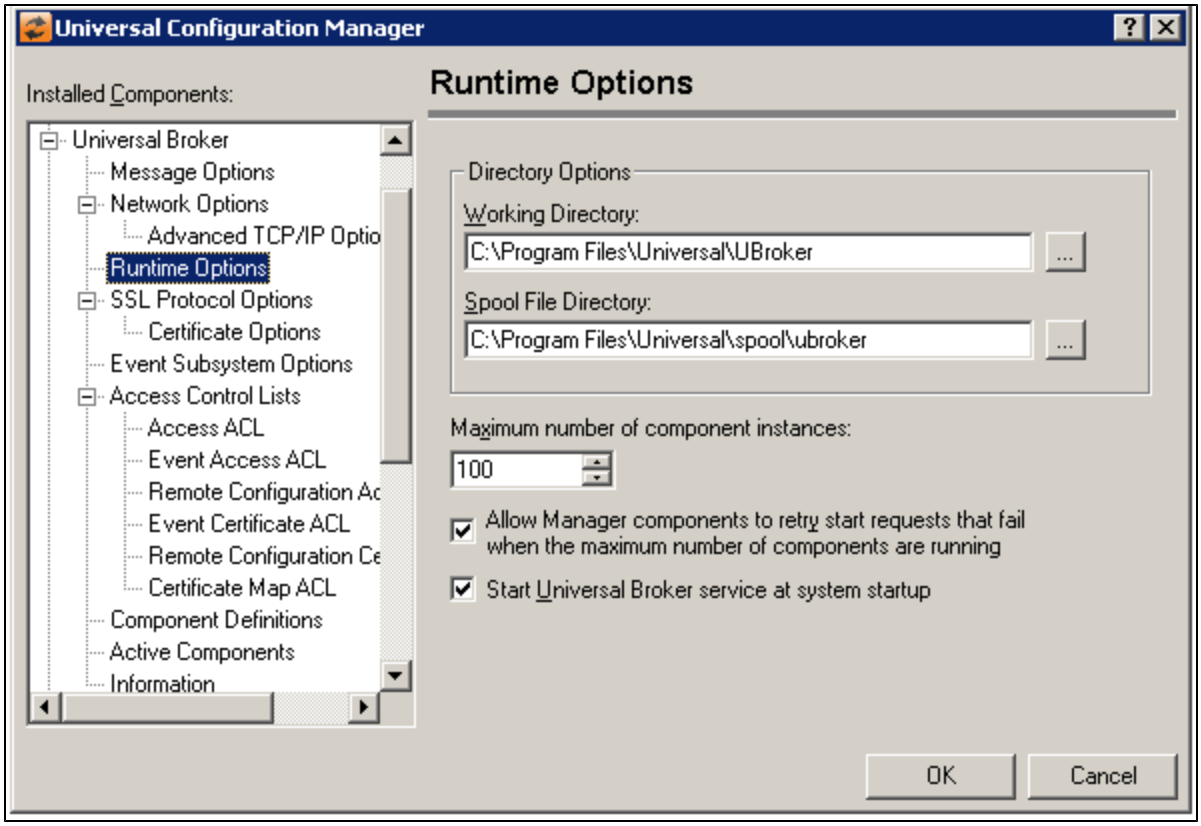


Field	Description
TCP/IP Options	**Multi-homed configuration** A multi-homed host is a machine that has more than one network interface card installed. On such machines, Universal Broker may be configured to accept incoming connections on only one of the interfaces. To specify this address, check the option and enter a valid host name or IP address in the space provided. Otherwise, leave the option unchecked, and Universal Broker will accept connections on all network interfaces. **DNS cache timeout** Specifies the number of seconds to retain a resolved host name in the Broker memory cache. A value of 0 specifies no caching.
Service Interface Options	**Port** Enter a port number or service name that will be used to accept incoming connections from Universal components. **Queue size** Specifies the number of connection requests that will be queued by Universal Broker. When the number of pending connection requests reaches this value, additional connection requests will be denied and a "connection refused" error will be issued. The default value for this option is 100.
Component Interface Options	**Port** Enter a port number or service name that will be used for communications with Universal components. **Queue size** Specifies the number of connection requests that will be queued by Universal Broker. When the number of pending connection requests reaches this value, additional connection requests will be denied and a "connection refused" error will be issued. The default value for this option is 100.
Code Page	Sets the translation table used to translate text-based data transmitted across the network.



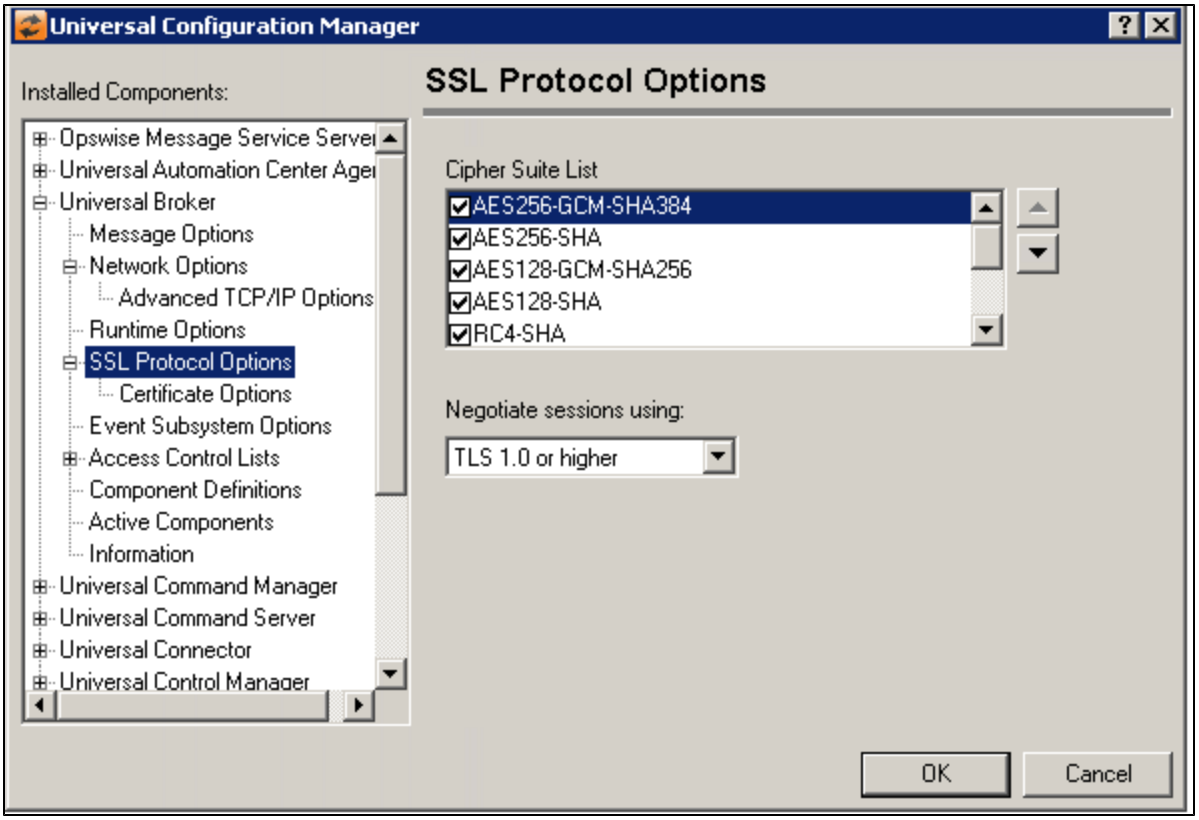
Field	Description
Override the default TCP/IP receive buffer and set to:	(No help provided.)
Override the default TCP/IP receive buffer and set to:	(No help provided.)

Universal Broker: Runtime Options

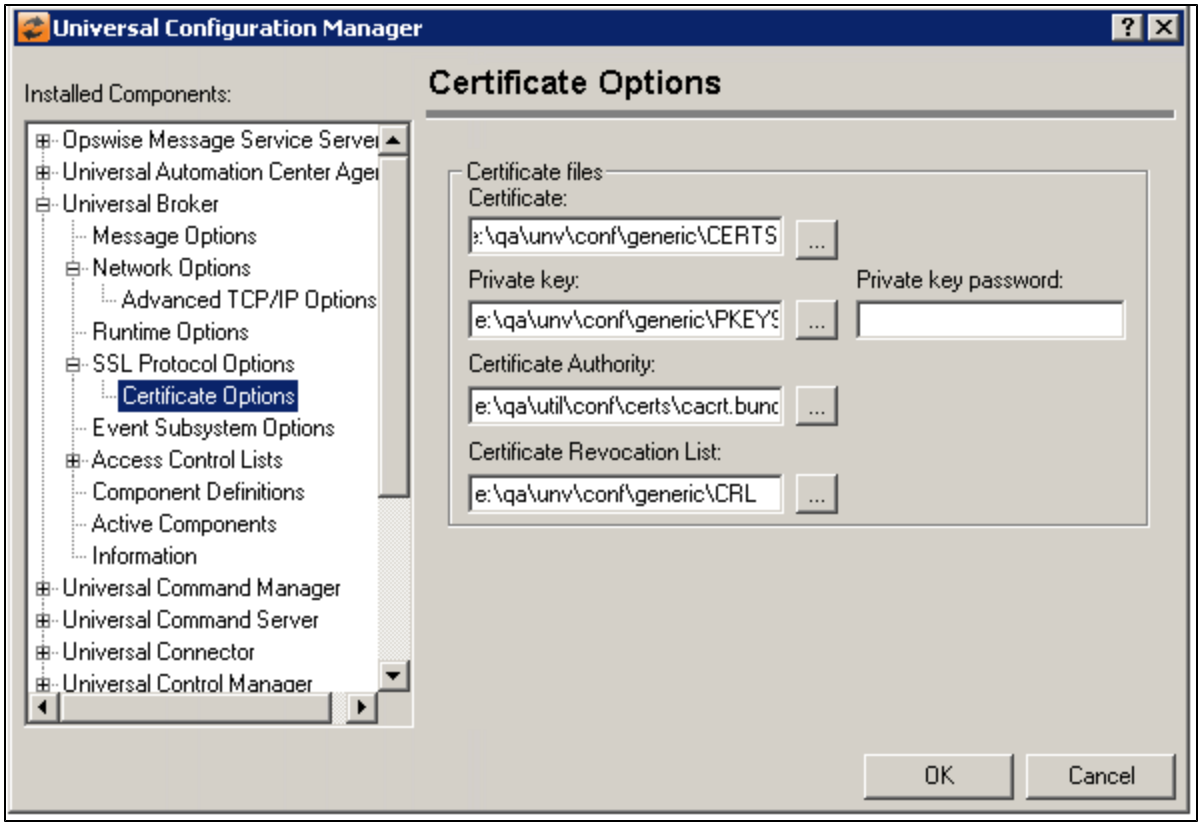


Field	Description
Directory Options	**Working Directory** Specifies the folder used by Universal Broker to store any generated output files. This folder will also be used as the default working directory for each component started by Universal Broker. **Spool File Directory** The directory where the spool files are stored. This directory must be located on local storage. It may not be located on any storage area accessed via the network which includes but is not limited to NFS, SAMBA, Microsoft File Sharing. **Start Universal Broker Service at system startup** Instructs the Universal Broker NT service to start when the system starts.
Max Number of Component Instances	Specifies the maximum number of Universal component instances that can be run at the same time.
Allow Manager components to retry start requests that fail win the max number of components are running.	(No help provided.)
Start Universal Broker Service at System Startup	Instructs the Universal Broker NT service to start when the system starts.

Universal Broker: SSL/TLS Protocol Options

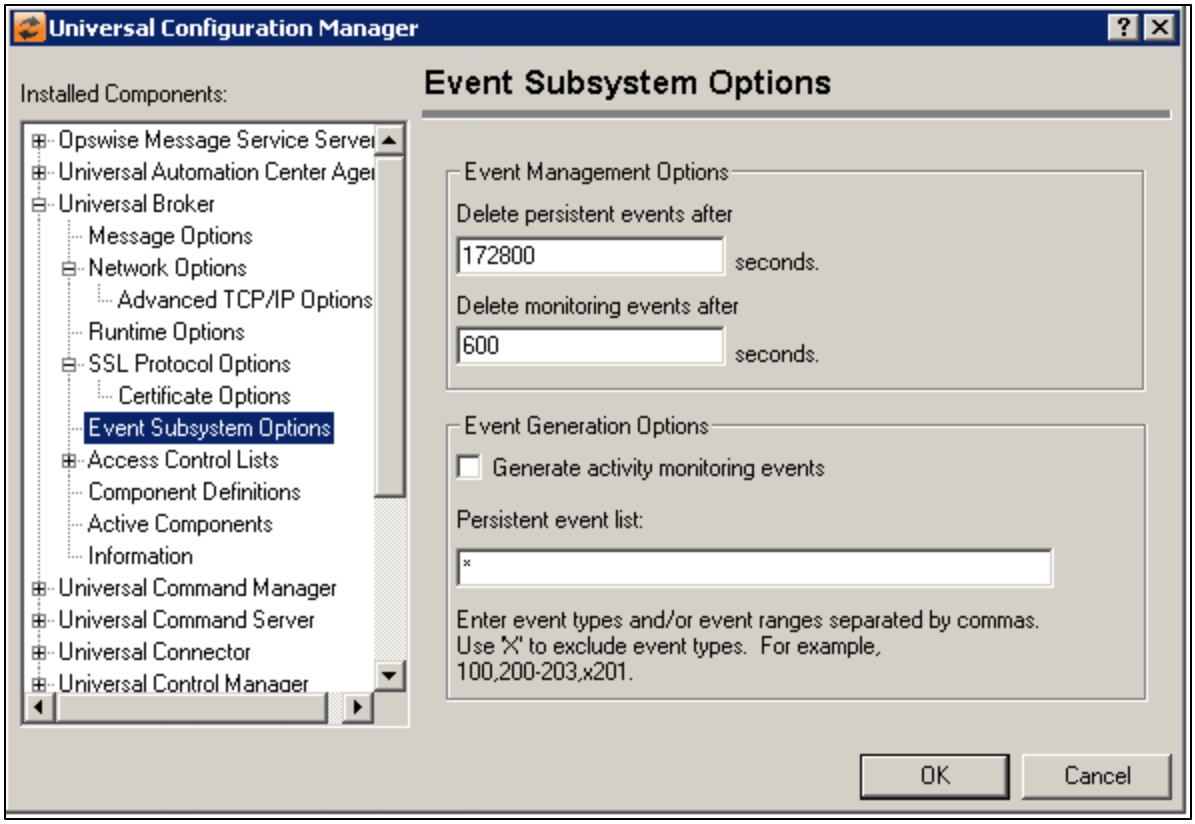


Field	Description
Cipher Suite List	The cipher suite specifies the symmetric and message digest algorithms used for a session. The cipher suite is negotiated as part of the SSL/TLS handshake. If one of the cipher suites in the list cannot be agreed upon, the SSL/TLS handshake will fail, and the connection will be closed.
Negotiate Sessions using	(No help provided.)



Field	Description
Certificate Files	**Certificate** The Certificate option specifies the X.509 certificate file that identifies the program to remote components with which it connects using the SSL/TLS protocol. **Private Key** The Private key option specifies the private key file associated with the X.509 certificate specified in the Certificate file. **Private Key Password** The Private key password specifies the password associated with the private key file. **Certificate Authority** The Certificate Authority option specifies a file with a list of trusted authorities. Any certificate received that is signed by a trusted Certificate Authority is accepted. **Certificate Revocation List** The Certificate Revocation List option specifies a file with a list of revoked certificates. Any certificate received that is in this list will not be accepted.

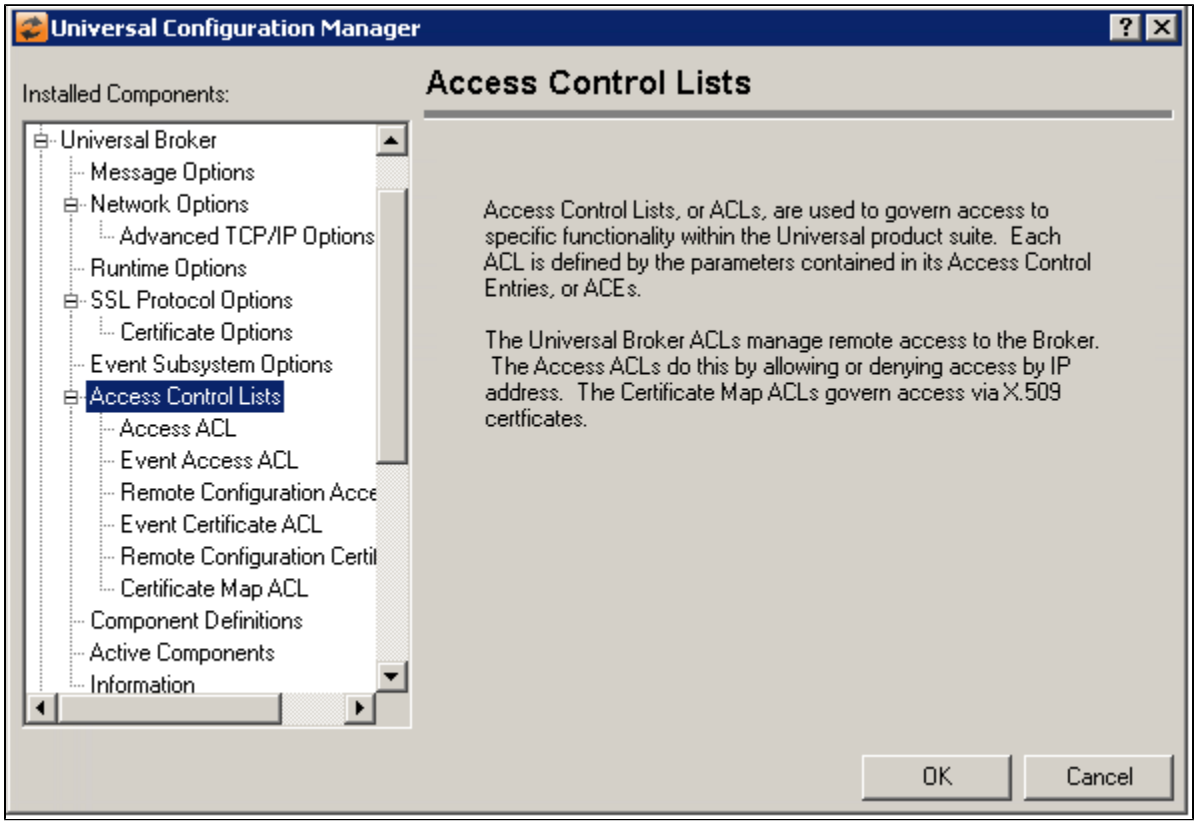
Universal Broker: Event Subsystem Options



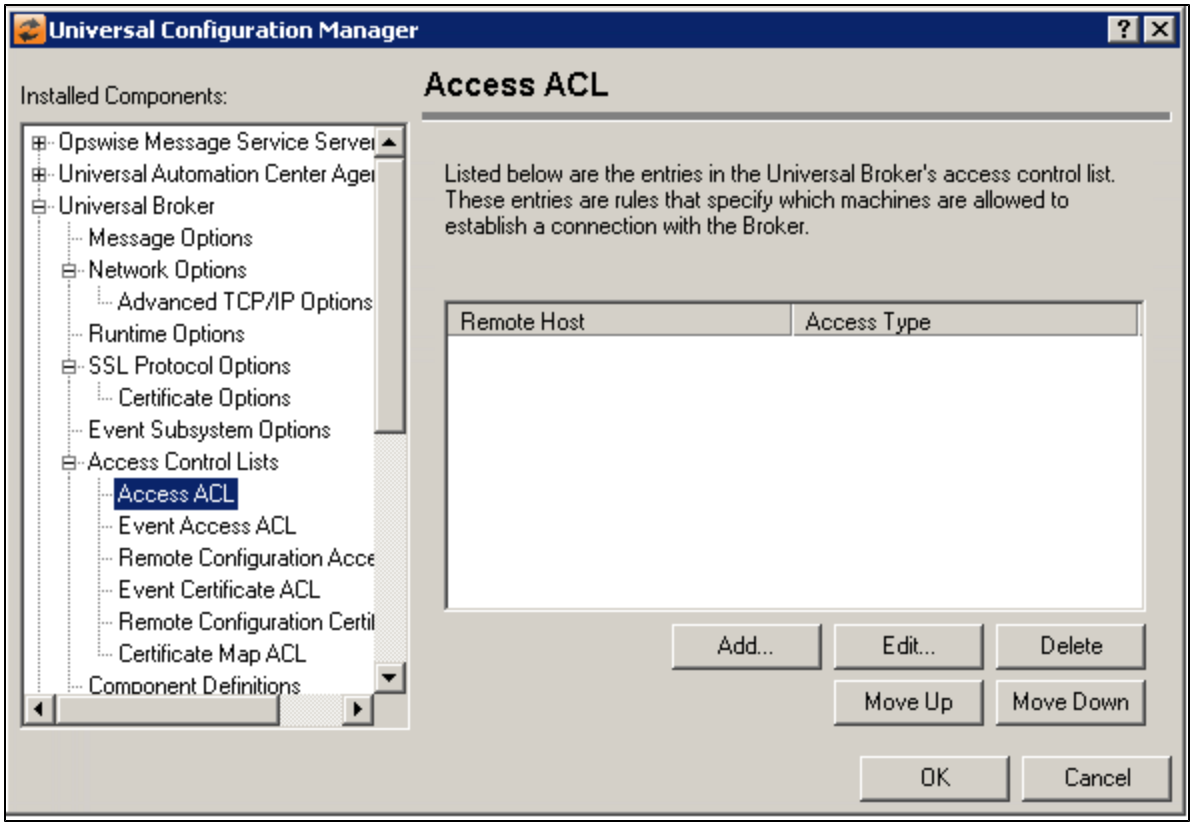
Field	Description
Event Management Options	**Delete Persistent events after** This option specifies how long, in seconds, an event message that is identified as a persistent event will wait to be received by a Universal Enterprise Controller (UEC). Persistent event messages are those whose types are contained in the event generation list. If a persistent event message is not delivered to UEC within the specified time, the Universal Broker will delete the event record from its local Universal Event Subsystem (UES) database. The default value for this option is 172800 seconds, which is equivalent to 2 days. **Delete Monitoring events after** This option sepcifies the maximum amount of time, in seconds, event messages used for product activity monitoring will remain in the Universal Broker's local Universal Event Subystem (UES) database. If a monitoring event message is not delivered to a Universal Enterprise Controller (UEC) within the specified time, the Broker will delete the record from its local UES database. The default value for this option is 600 seconds.
Event Generation Options	**Generate Activity monitoring events** When this option is selected, the Universal Broker will generate events that monitor product activity. This option is selected by default. **Persistent event list** Specifies which events are to be generated and processed as persistent events. A persistent event is saved in a Universal Enterprise Controller (UEC) event database for long-term storage. Enter a list and/or range of numeric event types, separated by a comma. To exclude an event or range of events, precede the entry with an upper- or lower-case 'X'. Use an asterisk (*) to specify all event types. For example: • x* Do not generate any events (this is the default) • * Generate all event types • 100 Generate all event types except 100 • 100,200-205 Generate event type 100 and 200 through 205 • 100,200-205,x202 Generate event types 100 and 200 thru 205, excluding event type 202

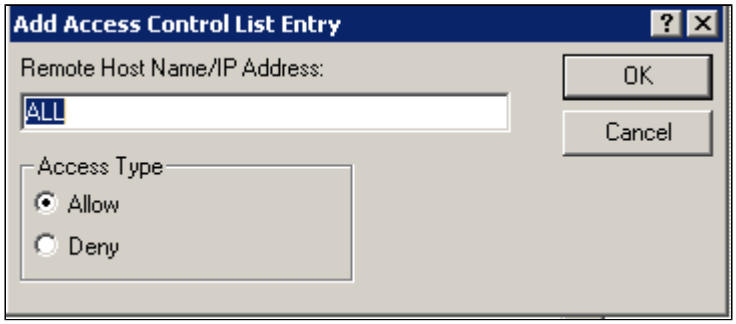
	Entries are processed from left to right, and the list is order dependent. For example, an entry of x*,100 prevents generation of all event types except 100, while an entry of 100,x* prevents generation of all event types (the right-most entry takes precedence).

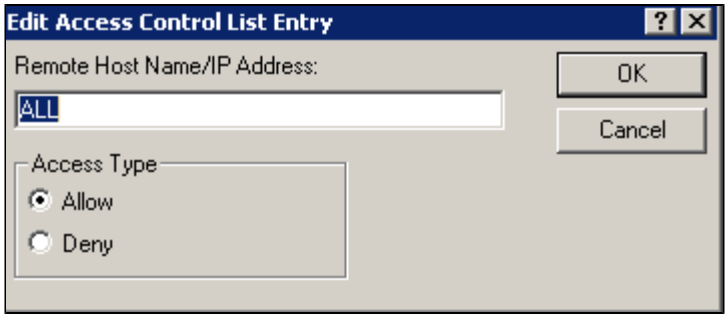
Universal Broker: Access Control Lists



Universal Broker: Access Control Lists - Access ACL



Field	Description
Access Control List	Displays the entries contained in the Universal Broker Access Control List (ACL). This list is used to grant or deny access to Universal Broker based on the remote system's IP address. The order in which ACL entries are evaluated and applied are based on their position in this list. Use the Move Up and Move Down buttons to change an entry's priority.
Add Button	Displays a dialog box that allows an entry to be added to the Access Control List. The fields in this dialog are populated with default values.  **Remote Host Name/IP Address** The IP address or name of the remote host for which access will be allowed or denied. To apply an access control entry to all IP addresses, use the keyword ALL (in all caps). **Access Type** Will cause the Universal Broker to accept/reject incoming connections from the specified host.
Edit Button	Displays a dialog that allows the selected Access Control List entry to be modified.

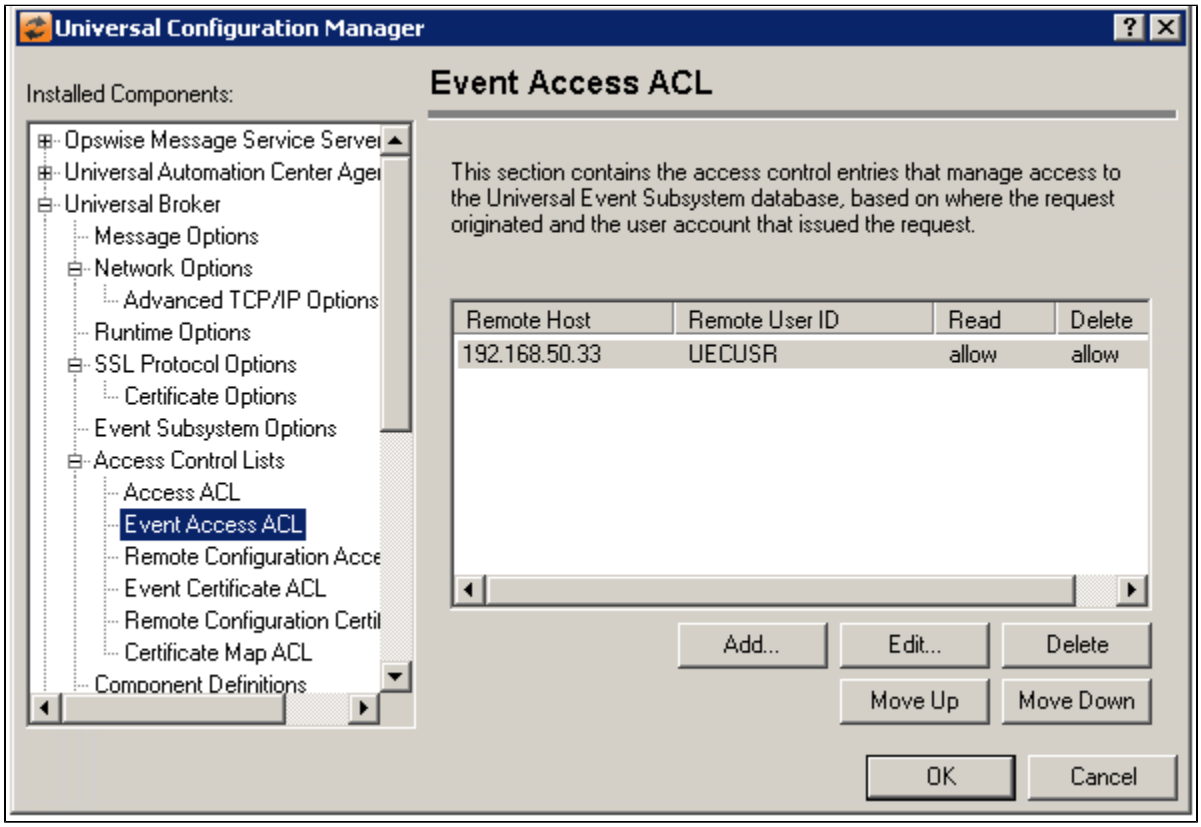


Remote Host Name/IP Address
The IP address or name of the remote host for which access will be allowed or denied. To apply an access control entry to all IP addresses, use the keyword ALL (in all caps).

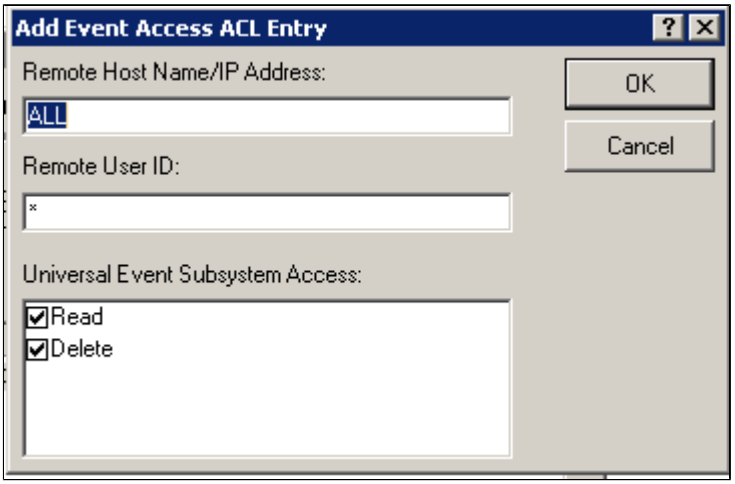
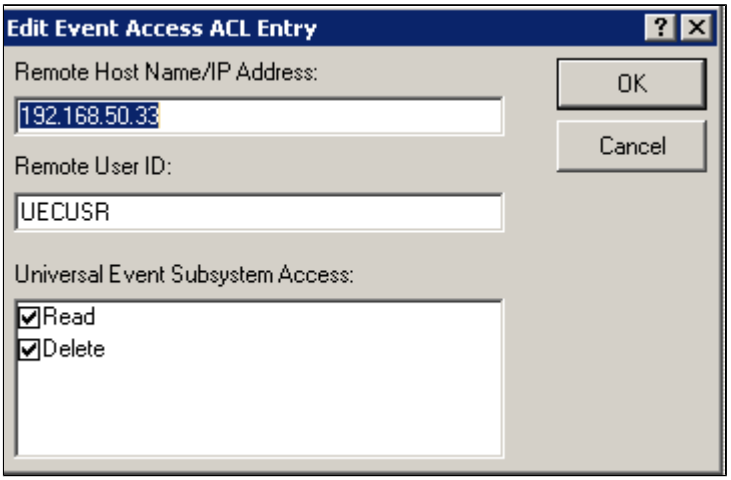
Access Type
Will cause the Universal Broker to accept/reject incoming connections from the specified host.

Delete Button	Removes the selected item from the Access Control List.
Move Up Button	Moves the selected item up one position in the list.
Move Down Button	Moves the selected item down one position in the list.

Universal Broker: Access Control Lists - Event Access ACL

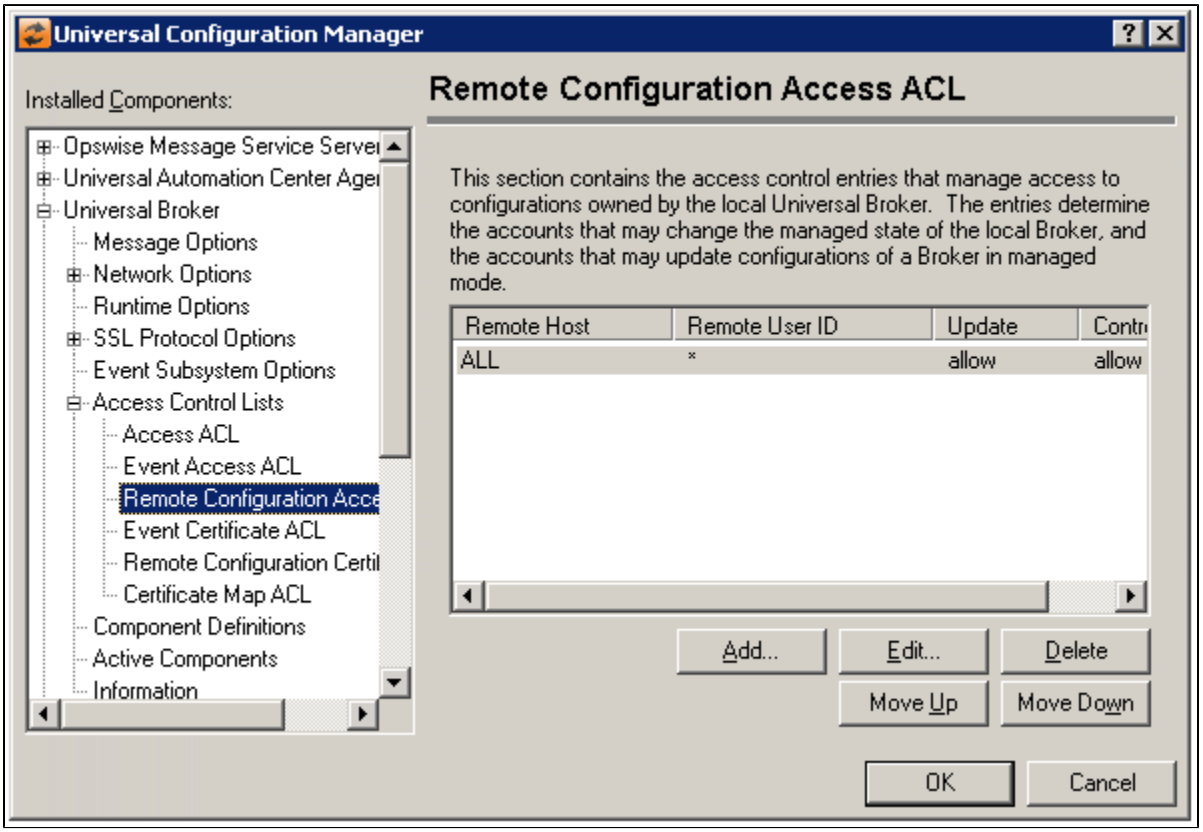


Field	Description
	Event Access ACL List Box Displays the Access Control List (ACL) entries that manage access to the Universal Event Subsystem (UES) database. The list is

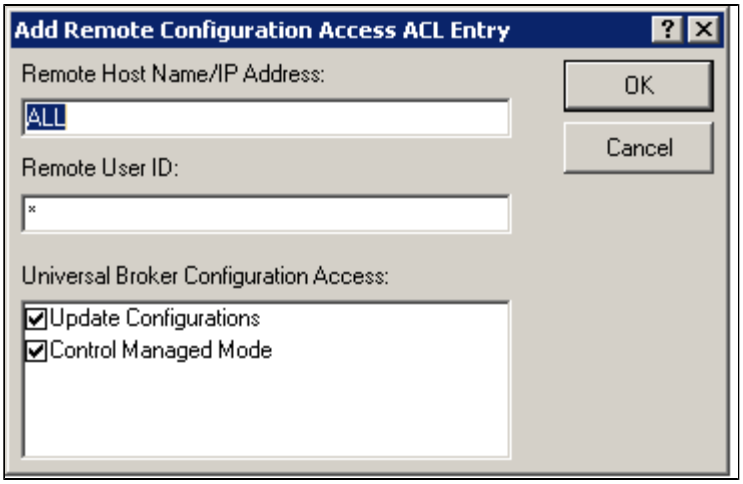
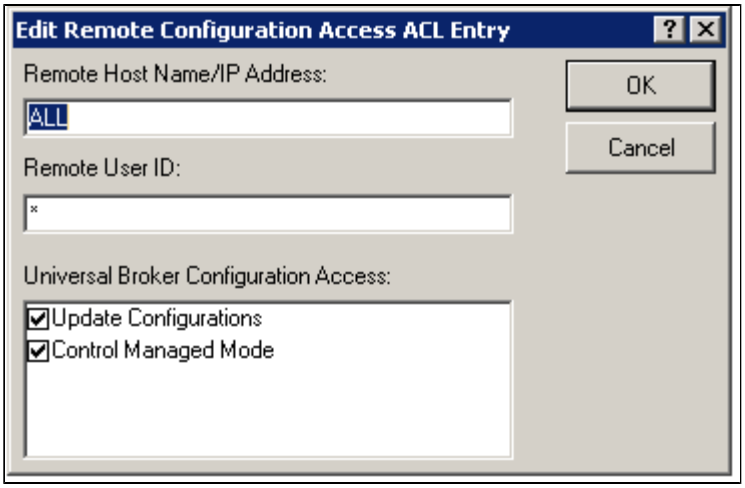
Event Access ACL	used to grant or deny read and/or delete access to the UES database, based on which system the request comes from and the user account that issued the request. The order in which ACL entries are evaluated and applied are based on their position in this list. Use the Move Up and Move Down buttons to change an entry's priority.
Add Button	Displays a dialog box that allows an entry to be added to the Access Control List. The fields in this dialog are populated with default values.  **Remove Host Name/IP Address** The IP address or name of the remote host for which access will be allowed or denied. To apply an access control entry to all IP addresses, use the keyword ALL (in all caps). **Remote User ID** The ID of a remote user account to which access will be allowed or denied. To apply an access control entry to user accounts, use an asterisk *. **Universal Event Subsystem Access** Displays the types of Universal Event Subsystem database access available. Check the box next to the access type to grant the specified access to the user. Uncheck the box to deny access to the user.
Edit Button	Displays a dialog that allows the selected Access Control List entry to be modified.  **Remove Host Name/IP Address** The IP address or name of the remote host for which access will be allowed or denied. To apply an access control entry to all IP addresses, use the keyword ALL (in all caps). **Remote User ID** The ID of a remote user account to which access will be allowed or denied. To apply an access control entry to user accounts, use an asterisk *.

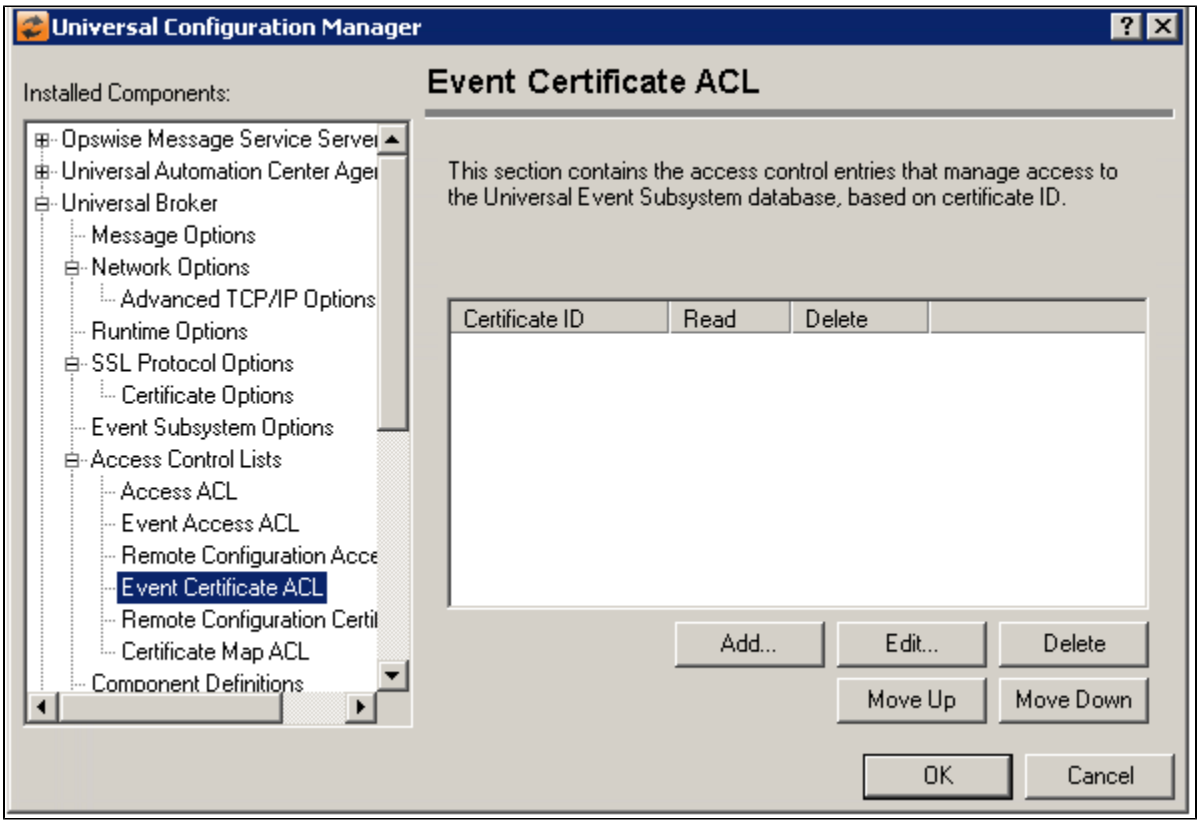
	Universal Event Subsystem Access Displays the types of Universal Event Subsystem database access available. Check the box next to the access type to grant the specified access to the user. Uncheck the box to deny access to the user.
Delete Button	Removes the selected item from the Access Control List.
Move Up Button	Moves the selected item up one position in the list.
Move Down Button	Moves the selected item down one position in the list.

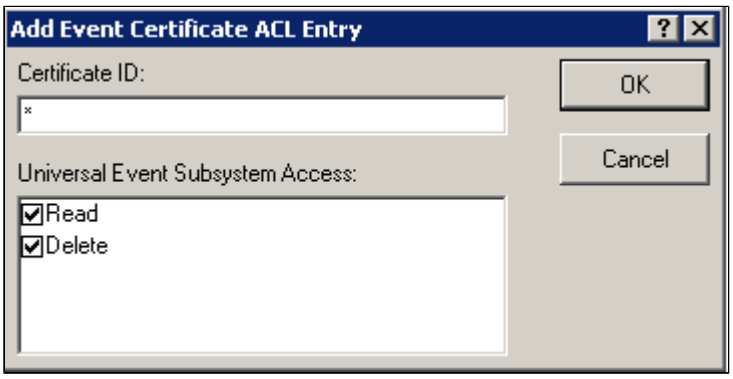
Universal Broker: Access Control Lists - Remote Configuration Access ACL

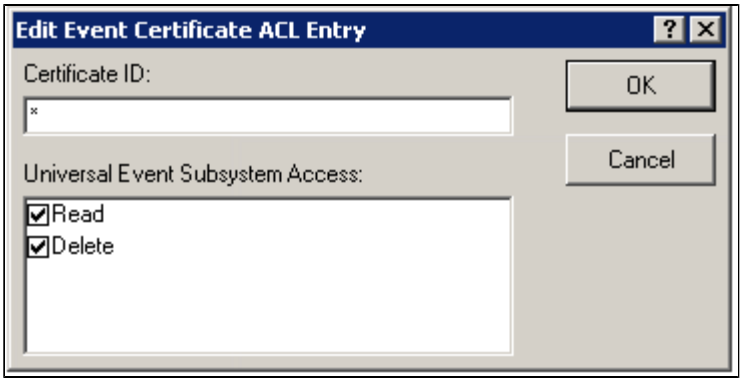


Field	Description
Remote Configuration Access ACL	**Remote Configuration Access ACL List Box** Displays the Access Control List (ACL) entries that manage requests to place the local Universal Broker into or out of managed mode, and requests to update Universal Product configurations when the local Broker is in managed mode. Requests are granted or denied based on the system from which the request was made, and the user account that issued the request. The order in which ACL entries are evaluated and applied are based on their position in this list. Use the Move Up and Move Down buttons to change an entry's priority.
Add Button	Displays a dialog box that allows an entry to be added to the Access Control List. The fields in this dialog are populated with default values.

	 **Remote HostName/IP Address** The IP address or name of the remote host for which access will be allowed or denied. To apply an access control entry to all IP addresses, use the keyword ALL (in all caps). **Remote User ID** The ID of a remote user account to which access will be allowed or denied. To apply an access control entry to user accounts, use an asterisk (*). **Universal Broker Configuration Access** Displays the types of Remote Configuration access available. Check the box next to the access type to grant the specified access to the user. Uncheck the box to deny access to the user.
Edit Button	Displays a dialog that allows the selected Access Control List entry to be modified.  **Certificate ID** If the value entered here is listed in the Certificate Map, the authority to execute the request is granted or denied. **Universal Broker Config Access** Displays the types of Remote Configuration access available. Check the box next to the access type to grant the specified access whenever the requestor presents the corresponding certificate. Uncheck the box to deny access.
Delete Button	Removes the selected item from the Access Control List.
Move Up Button	Moves the selected item up one position in the list.
Move Down Button	Moves the selected item down one position in the list.



Field	Description
Event Certificate ACL	**Event Certificate ACL List Box** Displays the Access Control List (ACL) entries that manage access to the Universal Event Subsystem (UES) database. The list is used to grant or deny read and/or delete access to the UES database, based on the ID of a certificate presented by the requestor. This certificate ID must have a corresponding entry in the Universal Broker Certificate Map ACL. The order in which ACL entries are evaluated and applied are based on their position in this list. Use the Move Up and Move Down buttons to change an entry's priority.
Add Button	Displays a dialog box that allows an entry to be added to the Access Control List. The fields in this dialog are populated with default values.  **Certificate ID** If the value entered here is listed in the Certificate Map, the authority to execute the request is granted or denied. **Universal Event Subsystem Access** Displays the types of Universal Event Subsystem database access available. Check the box next to the access type to grant the specified access whenever the requestor presents the corresponding certificate. Uncheck the box to deny access.
Edit Button	Displays a dialog that allows the selected Access Control List entry to be modified.

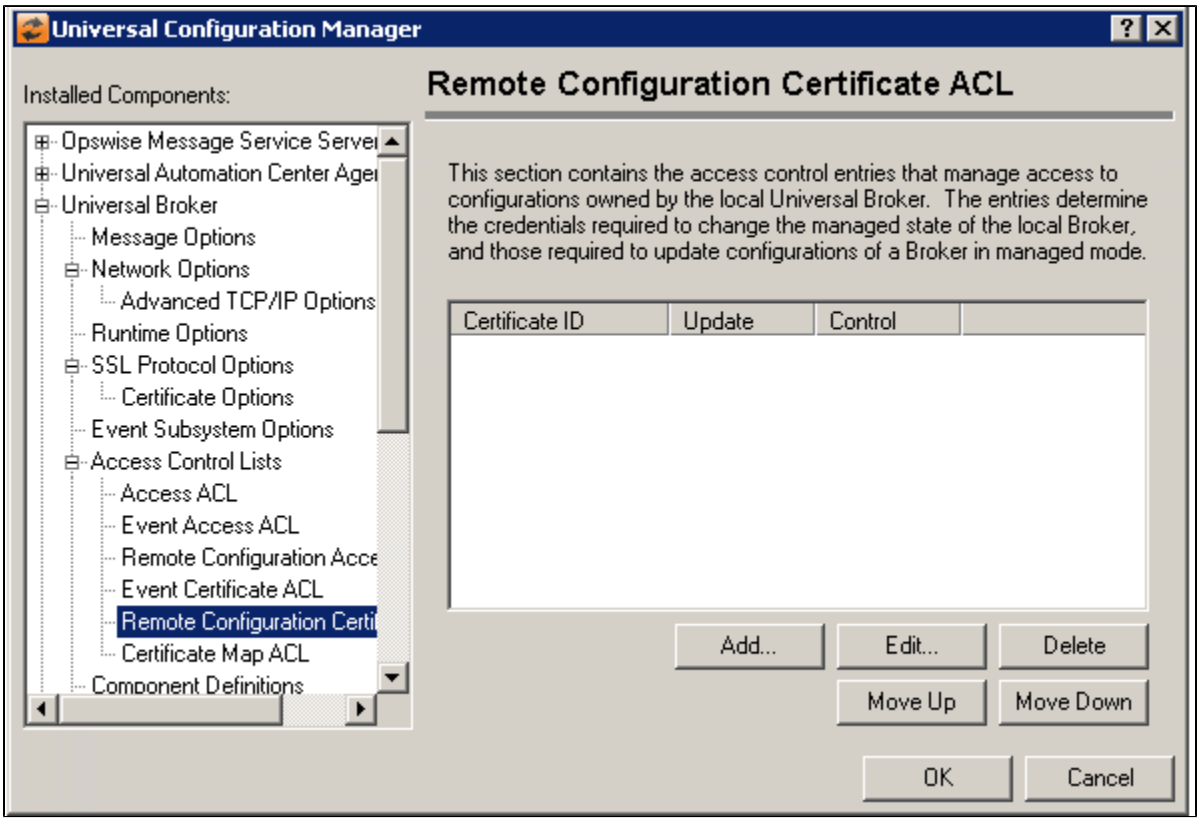


Certificate ID
If the value entered here is listed in the Certificate Map, the authority to execute the request is granted or denied.

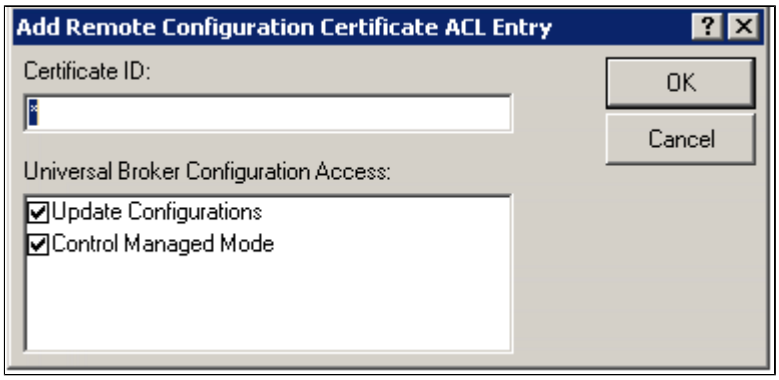
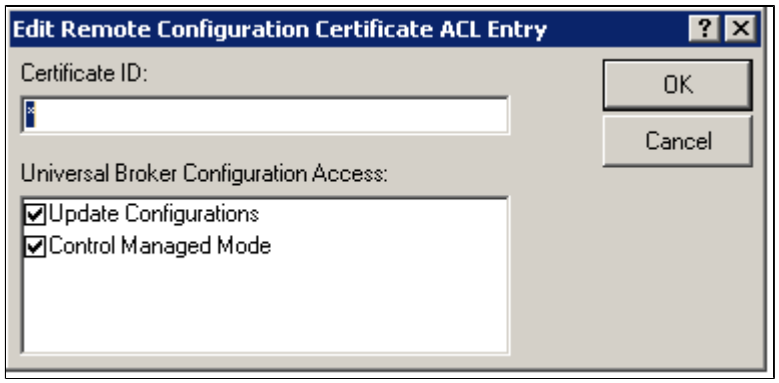
Universal Event Subsystem Access
Displays the types of Universal Event Subsystem database access available. Check the box next to the access type to grant the specified access whenever the requestor presents the corresponding certificate. Uncheck the box to deny access.

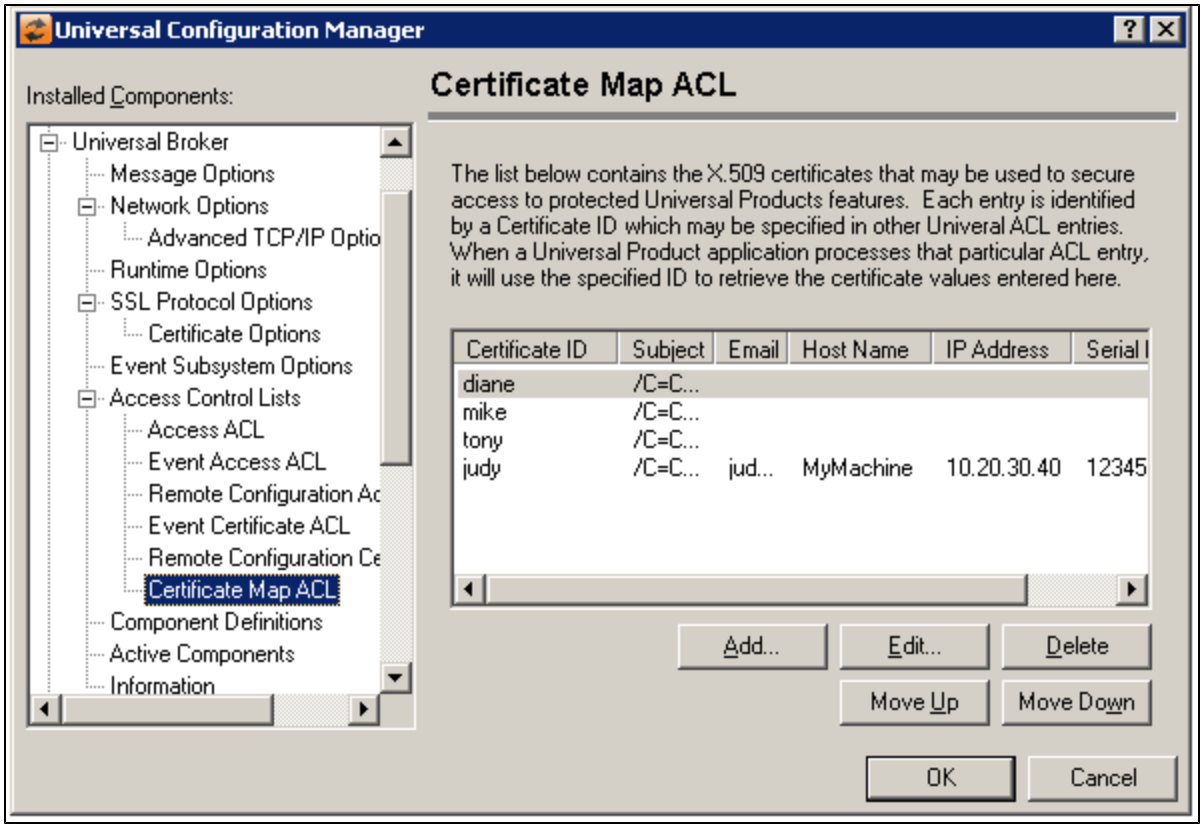
Delete Button	Removes the selected item from the Access Control List.
Move Up Button	Moves the selected item up one position in the list.
Move Down Button	Moves the selected item down one position in the list.

Universal Broker: Access Control Lists - Remote Configuration Certificate ACL

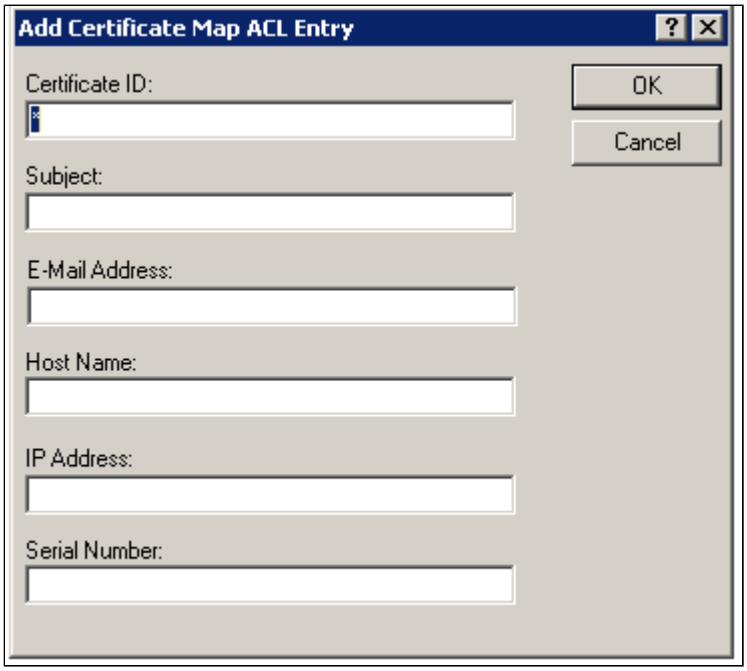


Field	Description

Remote Configuration Certificate ACL	**Remote Configuration Certificate ACL List Box** Displays the Access Control List (ACL) entries that manage requests to place the local Universal Broker into or out of managed mode, and requests to update Universal Product configurations when the local Broker is in managed mode. Requests are granted or denied based on the credentials a requestor presents in the specified certificate. This certificate must have a corresponding entry in the Universal Broker Certificate Map ACL. The order in which ACL entries are evaluated and applied are based on their position in this list. Use the Move Up and Move Down buttons to change an entry's priority.
Add Button	Displays a dialog box that allows an entry to be added to the Access Control List. The fields in this dialog are populated with default values.  **Certificate ID** If the value entered here is listed in the Certificate Map, the authority to execute the request is granted or denied. **Universal Broker Configuration Access** Displays the types of Remote Configuration access available. Check the box next to the access type to grant the specified access whenever the requestor presents the corresponding certificate. Uncheck the box to deny access.
Edit Button	Displays a dialog that allows the selected Access Control List entry to be modified.  **Certificate ID** If the value entered here is listed in the Certificate Map, the authority to execute the request is granted or denied. **Universal Broker Configuration Access** Displays the types of Remote Configuration access available. Check the box next to the access type to grant the specified access whenever the requestor presents the corresponding certificate. Uncheck the box to deny access.
Delete Button	Removes the selected item from the Access Control List.
Move Up Button	Moves the selected item up one position in the list.
Move Down Button	Moves the selected item down one position in the list.



Field	Description
Certificate Map ACL List Box	Displays the entries contained in the X.509 certificate mapping. The order in which entries are evaluated and applied are based on their position in this list. Use the Move Up and Move Down buttons to change an entry's priority.
Add Button	Displays a dialog box that allows an entry to be added to the Access Control List. The fields in this dialog are populated with default values.



Certificate ID
The value entered here is used as a key to establish a mapping between this entry and a Universal ACL entry that may reference it.

Subject
The value entered here is used to match the subject field of an installed X.509 certificate. The subject contains properties of the entity identified by the certificate. It is formatted as an X.501 Distinguished Name (DN), which is a hierarchical list of attributes referred to as Relative Distinguished Names (RDNs).

Some of the more common RDN values that may be used in a DN include:

- C
  Country Name
- CN
  Common Name
- L
  Locality
- O
  Organization
- OU
  Organizational Unit
- ST
  State

The following are examples of Distinguished Names:

- /C=US/ST=Georgia/L=Atlanta/O=Stonebranch, Inc./OU=R & D/CN=Joe Simson/
- /C=US/ST=Georgia/L=Atlanta/O=Stonebranch, Inc./OU=Operations/
- /C=US/ST=Georgia/L=Atlanta/O=Stonebranch, Inc./OU=Operations/CN=UEC/

E-Mail Address
The value entered here is used to match an installed X.509 certificate by comparing the dNSName of the subjectAltName extension value or the commonName (CN) Relative Distinguished Name attribute of a Distinguished Name's subject field. The host name entered here may contain wildcards. For example, *.acme.com.

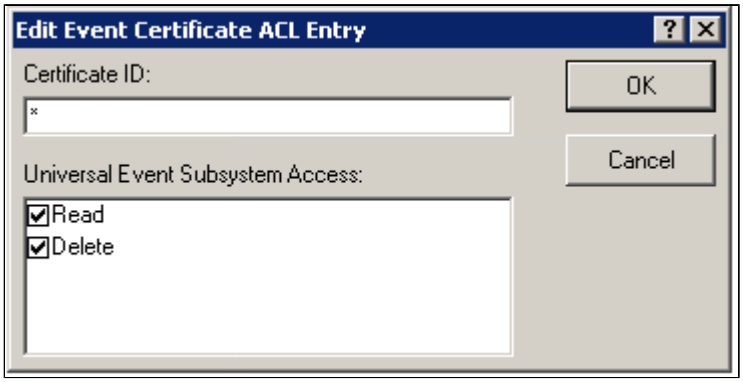
Host Name
The value entered here is used to match an installed X.509 certificate by comparing the dNSName of the subjectAltName extension value or the commonName (CN) Relative Distinguished Name attribute of a Distinguished Name's subject field. The host name entered here may contain wildcards. For example, *.acme.com.

IP Address
The value entered here is used to match an installed X.509 certificate by comparing the iPAddress of the subjectAltName extension value. The IP Address entered must be in dotted decimal format (i.e., xx.xx.xx.xx).

Serial Number
The value entered here is used to match the Serial Number of an installed X.509 certificate. Please note that serial numbers are not unique across all certificates, only for those certificates provided by specific Certificate Authorities (CA's).

Edit Button

Displays a dialog that allows the selected Access Control List entry to be modified.

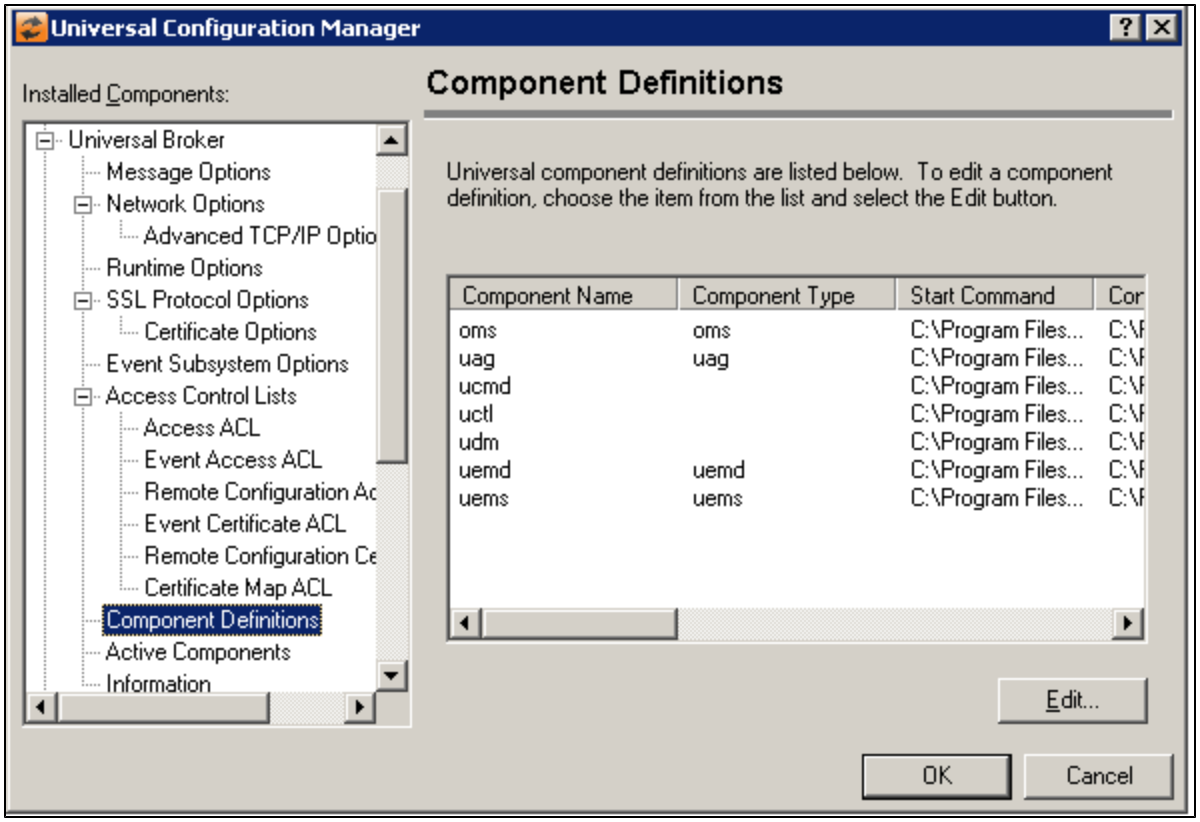


Certificate ID
The value entered here is used as a key to establish a mapping between this entry and a Universal ACL entry that may reference it.

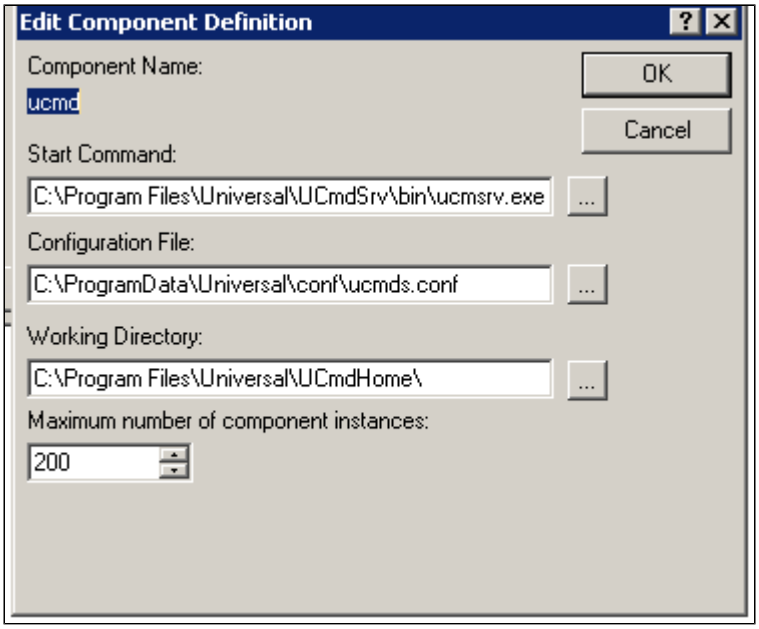
Subject
The value entered here is used to match the subject field of an installed X.509 certificate. The subject contains properties of the entity identified by the certificate. It is formatted as an X.501 Distinguished Name (DN), which is a hierarchical list of attributes

	referred to as Relative Distinguished Names (RDNs). Some of the more common RDN values that may be used in a DN include: • C Country Name • CN Common Name • L Locality • O Organization • OU Organizational Unit • ST State The following are examples of Distinguished Names: • /C=US/ST=Georgia/L=Atlanta/O=Stonebranch, Inc./OU=R & D/CN=Joe Simson/ • /C=US/ST=Georgia/L=Atlanta/O=Stonebranch, Inc./OU=Operations/ • /C=US/ST=Georgia/L=Atlanta/O=Stonebranch, Inc./OU=Operations/CN=UEC/ **E-Mail Address** The value entered here is used to match an installed X.509 certificate by comparing the dNSName of the subjectAltName extension value or the commonName (CN) Relative Distinguished Name attribute of a Distinguished Name's subject field. The host name entered here may contain wildcards. For example, *.acme.com. **Host Name** The value entered here is used to match an installed X.509 certificate by comparing the dNSName of the subjectAltName extension value or the commonName (CN) Relative Distinguished Name attribute of a Distinguished Name's subject field. The host name entered here may contain wildcards. For example, *.acme.com. **IP Address** The value entered here is used to match an installed X.509 certificate by comparing the iPAddress of the subjectAltName extension value. The IP Address entered must be in dotted decimal format (i.e., xx.xx.xx.xx). **Serial Number** The value entered here is used to match the Serial Number of an installed X.509 certificate. Please note that serial numbers are not unique across all certificates, only for those certificates provided by specific Certificate Authorities (CA's).
Delete Button	Removes the selected item from the Access Control List.
Move Up Button	Moves the selected item up one position in the list.
Move Down Button	Moves the selected item down one position in the list.

Universal Broker: Component Definitions



Field	Description
Universal Component Definitions List	Contains a list of definitions used by Universal Broker to manage Universal components.
Edit Button	Displays a dialog box that is used to modify an existing component definition.



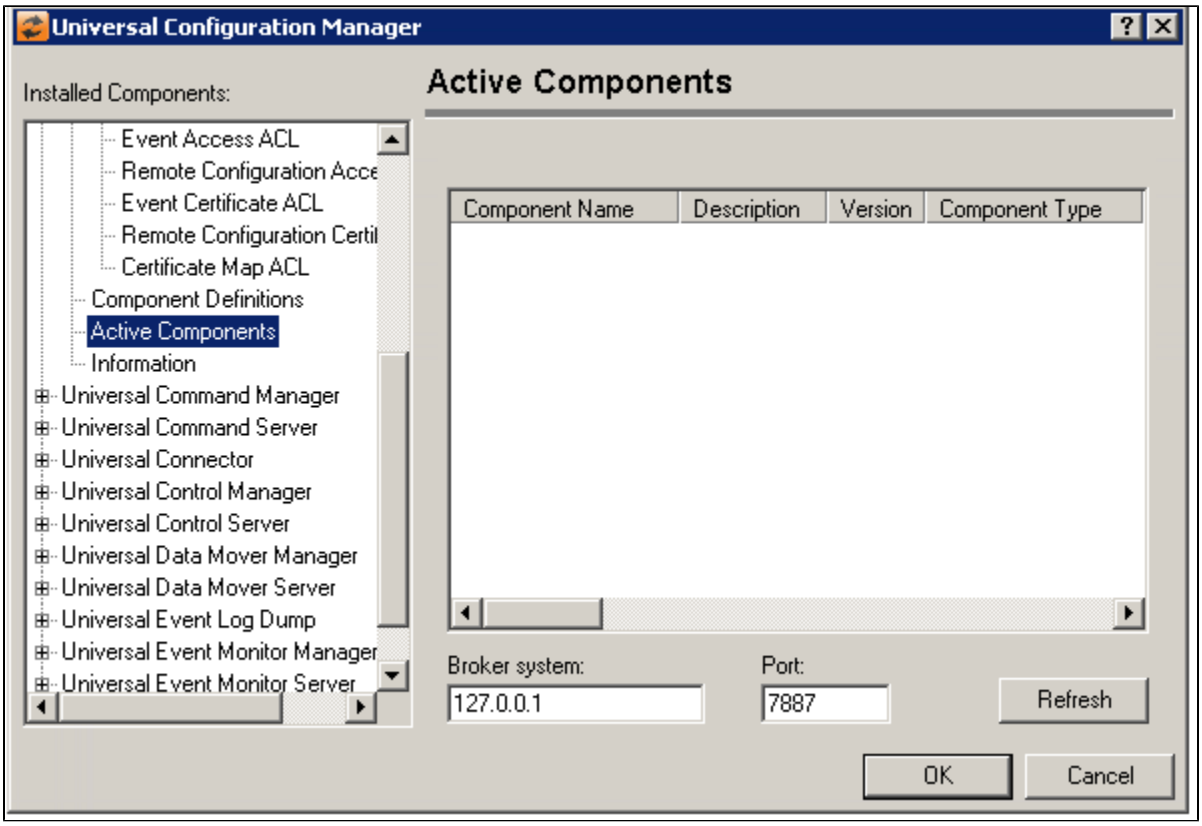
Start Command
The complete path to the component's executable file.

Configuration File
The name of the file in which configuration options for the component are stored.

Working Directory

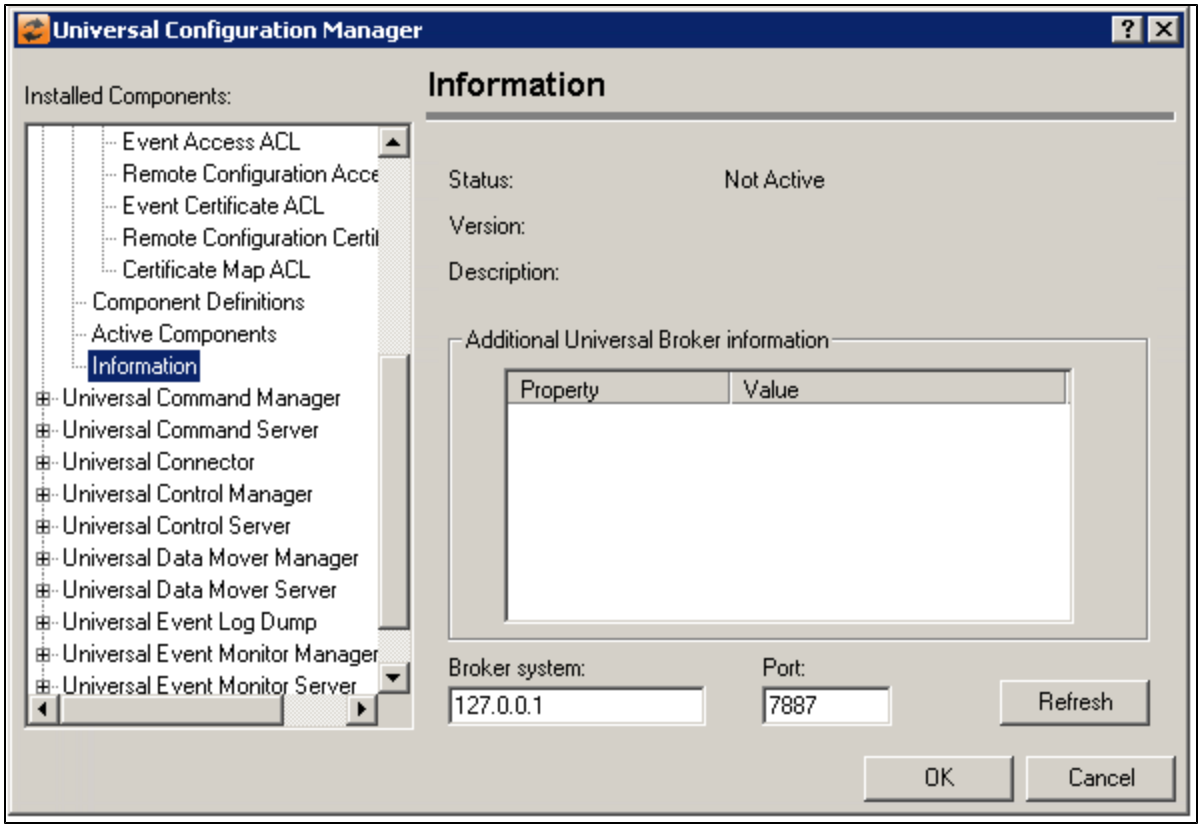
	The component's working folder. The component uses this folder to store any output files or log files. **Maximum number of component instances** The maximum number of components that are allowed to run simultaneously. Setting this value to 0 prevents the component from running at all.

Universal Broker: Active Components



Field	Description
Active Components List	Contains a list of components that are currently running.
Broker System	Address of the Broker to query.
Port	Port the Broker being queried is listening on.
Refresh Button	Updates the list of active components.

Universal Broker: Information

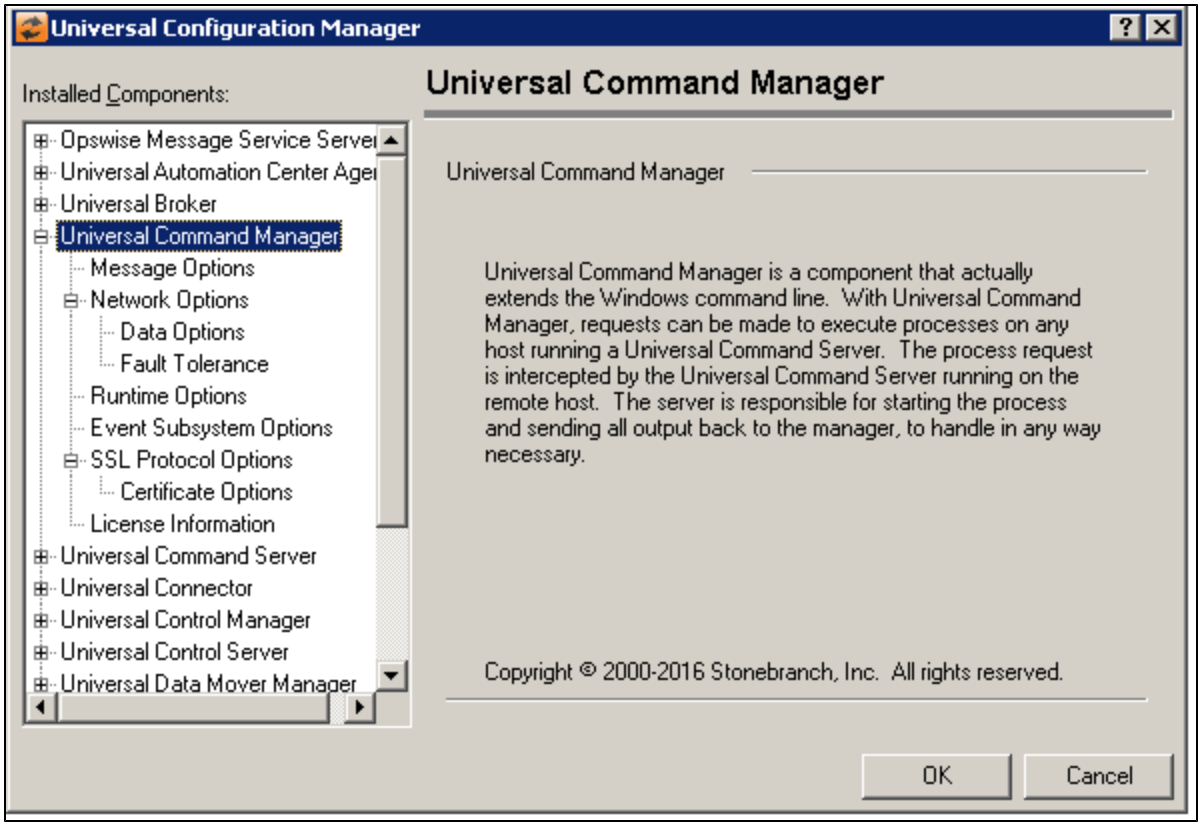


Field	Description
Status	Universal Broker status.
Version	Universal Broker version.
Description	Universal Broker description.
Additional Universal Broker Information	Displays properties of Universal Broker.
Broker System	Address of the Broker to query.
Port	Port the Broker being queried is listening on.
Refresh Button	Updates the broker information.

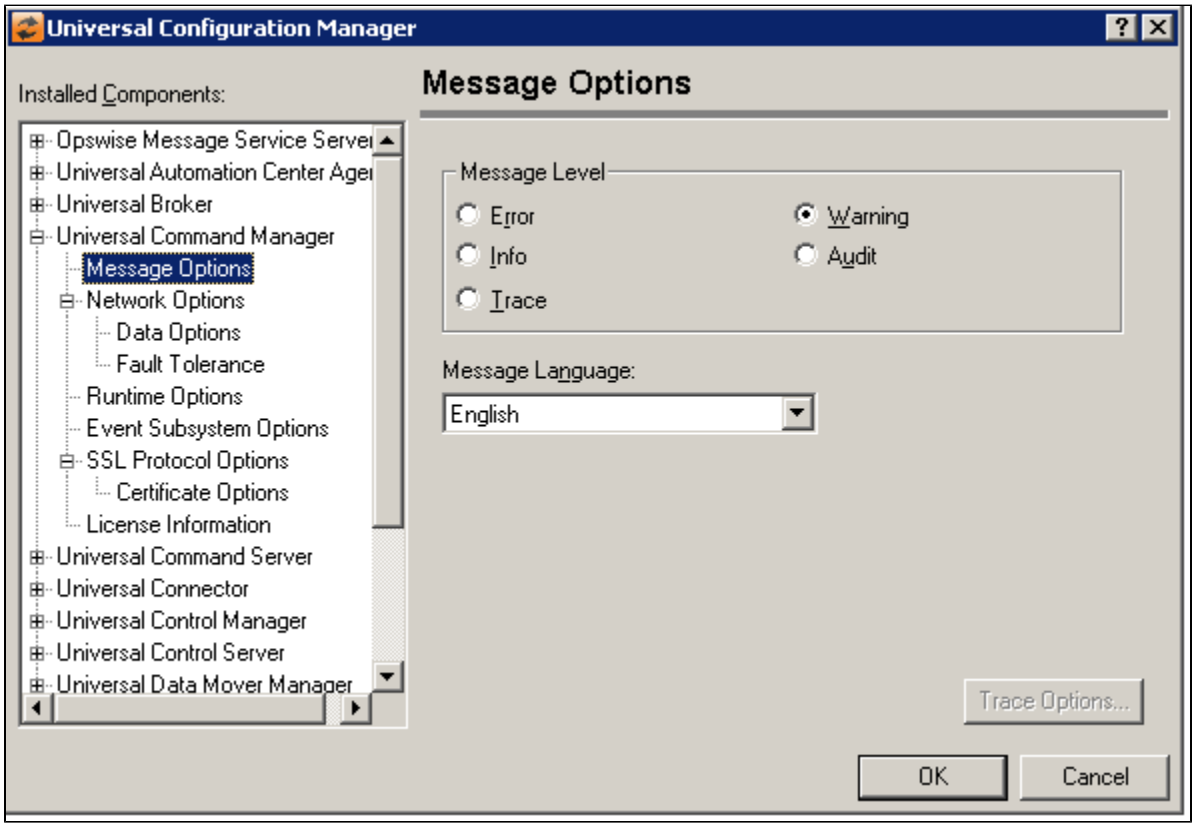
Universal Configuration Manager - Universal Command Manager

- Universal Command Manager
- Universal Command Manager: Message Options
- Universal Command Manager: Network Options
  - Universal Command Manager: Network Options - Data Options
  - Universal Command Manager: Network Options - Fault Tolerance
- Universal Command Manager: Runtime Options
- Universal Command Manager: Event Subsystem Options
- Universal Command Manager: SSL/TLS Protocol Options
  - Universal Command Manager: SSL Protocol Options - Certificate Options
- Universal Command Manager: License Information

Universal Command Manager



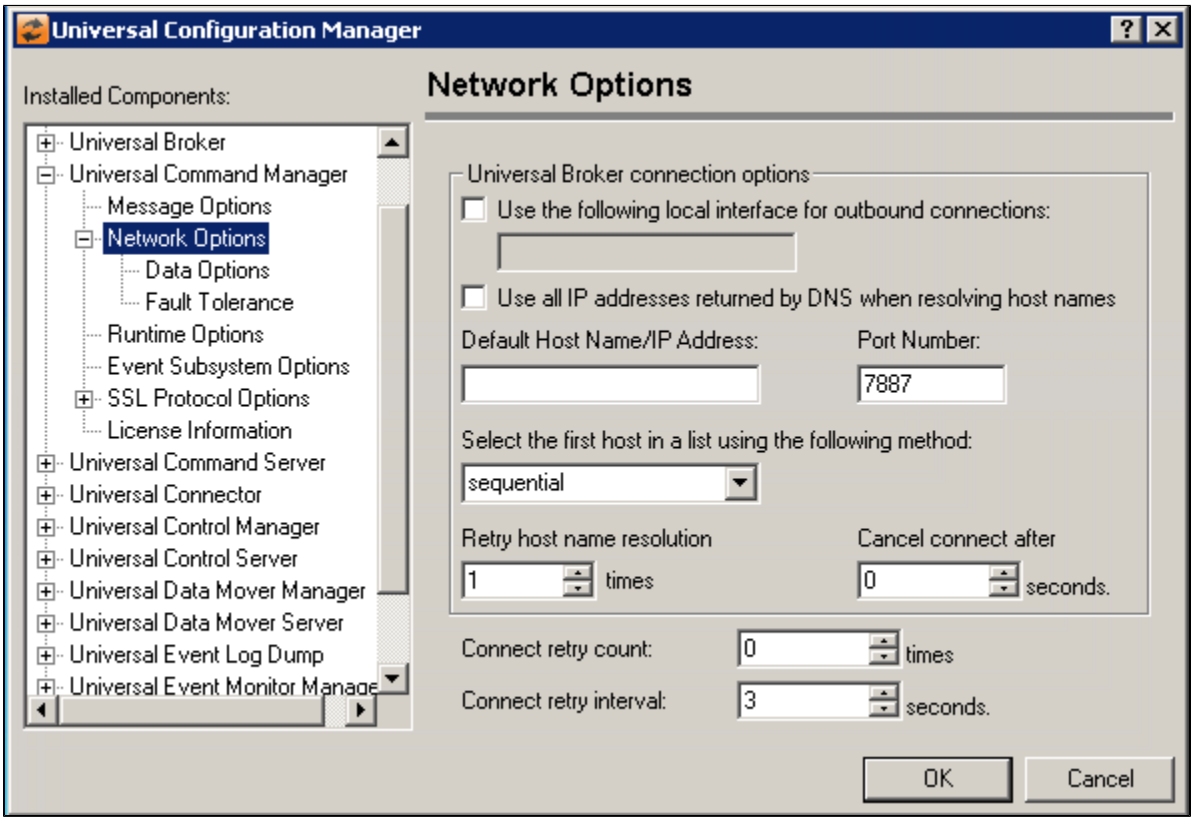
Universal Command Manager: Message Options



Field	Description
Message Level	

	Error Turns on message logging for errors only. **Warning** Turns on message logging for warnings and errors. **Info** Turns on message logging for all informational, warning and error messages. **Audit** Turns on message logging for all audit, informational, warning and error messages. **Trace** Allows all messages to be logged, including those used for debugging purposes.
Message Language	Specifies the language used to display error messages.

Universal Command Manager: Network Options

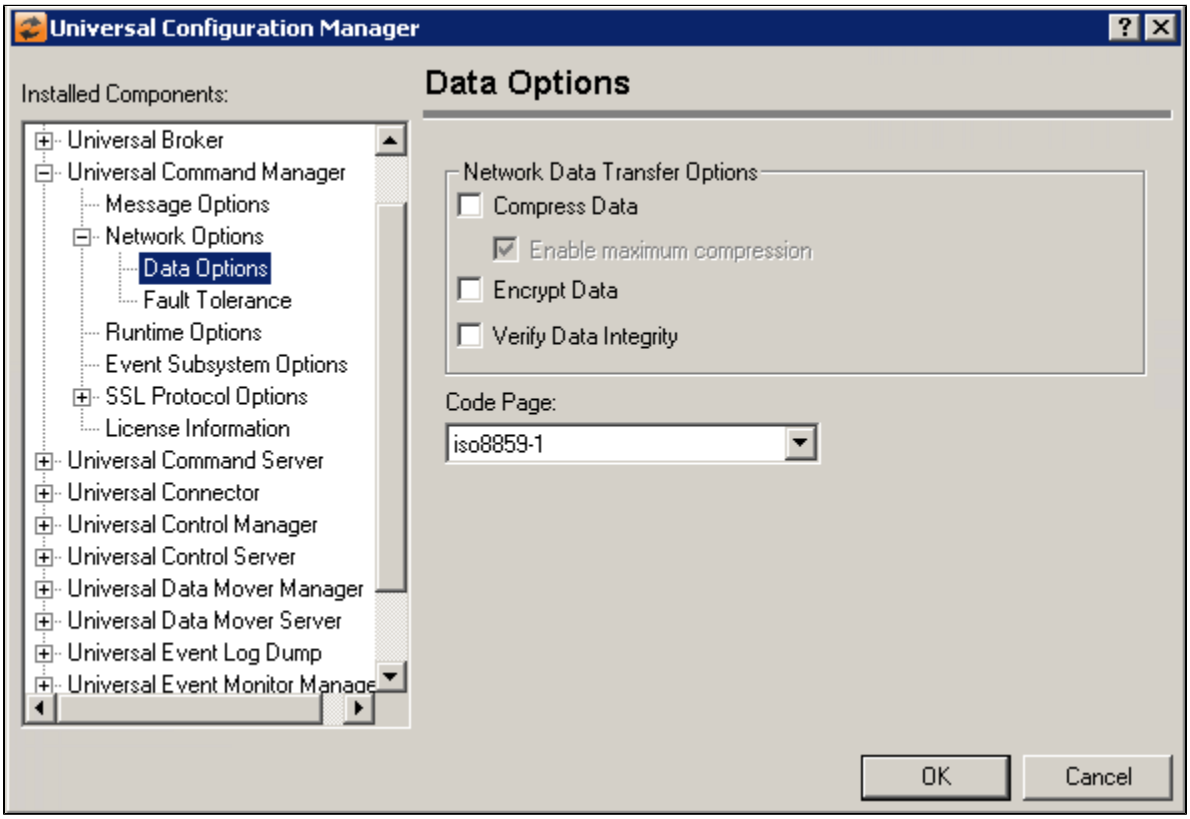


Field	Description
Universal Broker connection options	**Use the following local interface for outbound connections** When this option is selected, the IP address or alias of a local interface may be specified that will be used whenever UCMD Manager connects to a Universal Broker. If this option is not selected, any available local interface will be used. When more than one local interface is available, this option may be used to specify the IP address or host name that should be used whenever UCMD Manager connects to a Universal Broker. To have UCMD Manager use any local interface available, leave this value blank. **Use all IP addresses returned by DNS when resolving hostnames** When this option is selected, all IP addresses returned by a DNS query for a given host name are expanded in the list of hosts. When this option is not selected, only the first IP address for the resolved host name is used. This option is not selected by default. **Default Host Name/IP Address** Enter one or more valid IP addresses and/or host names – separated by a comma – to use as the default system(s) for process execution. A specific port number may also be specified for any one of the hosts in the list. The format for this list of one or more hosts is: host1[port1,host2port2,...,hostnportn] To run processes on the local system, enter a host name of localhost, or an IP address of 127.0.0.1.

	Port Number Enter a port number or service name that will be used to connect to Universal Broker. **Select the first host in a list using the following method** This option specifies how entries within a list of hosts are accessed as the Universal Command Manager attempts to establish a connection with a Universal Broker. When the sequential selection method is chosen, connection attempts begin with the first host in the list. Connection attempts continue with each subsequent host in the list, until a connection to a Universal Broker is established or until all hosts in the list have been tried. When the random method is chosen, connection attempts begin with a randomly-selected host. Connection attempts continue with each host in the list until a connection is established or until the first host selected is reached again. The default value for this option is sequential. **Retry host name resolution** Indicates the number of attempts that will be made to resolve a host name specified with the -host option on the Universal Command command line. This parameter can help resolve intermittent "host not found" errors. **Cancel connect after** This option specifies how long the Universal Command Manager will wait for each connection attempt to a Universal Broker to complete before moving on to the next host in the list. If there are no more hosts in the list, or connection attempts to all the hosts time out, the Manager will terminate. The default value for this option is 0 (zero), which instructs the UCMD Manager to use the default TCP/IP timeout value. This results in a blocking connect attempt, and is provided to perserve compatibility with pre-v320 behavior.
Connect retry count	This option specifies the number of times the UCMD Manager will try to establish a connection with a remote Broker. The Manager will continue to try each host in the list until it successfully connects to a remote Broker or until the number of connection attempts exceeds the value specified for this option. The default value for this option is 0 (zero), which instructs the UCMD Manager to only attempt a connection once for each host in the lis
Connect retry interval	This option specifies how long (in seconds) the UCMD Manager will wait before it retries an attempt to establish a connection with a remote Broker on a specified host. This option is only used when connect_retry_count is greater than 0 (zero). The default value for this option is 3.

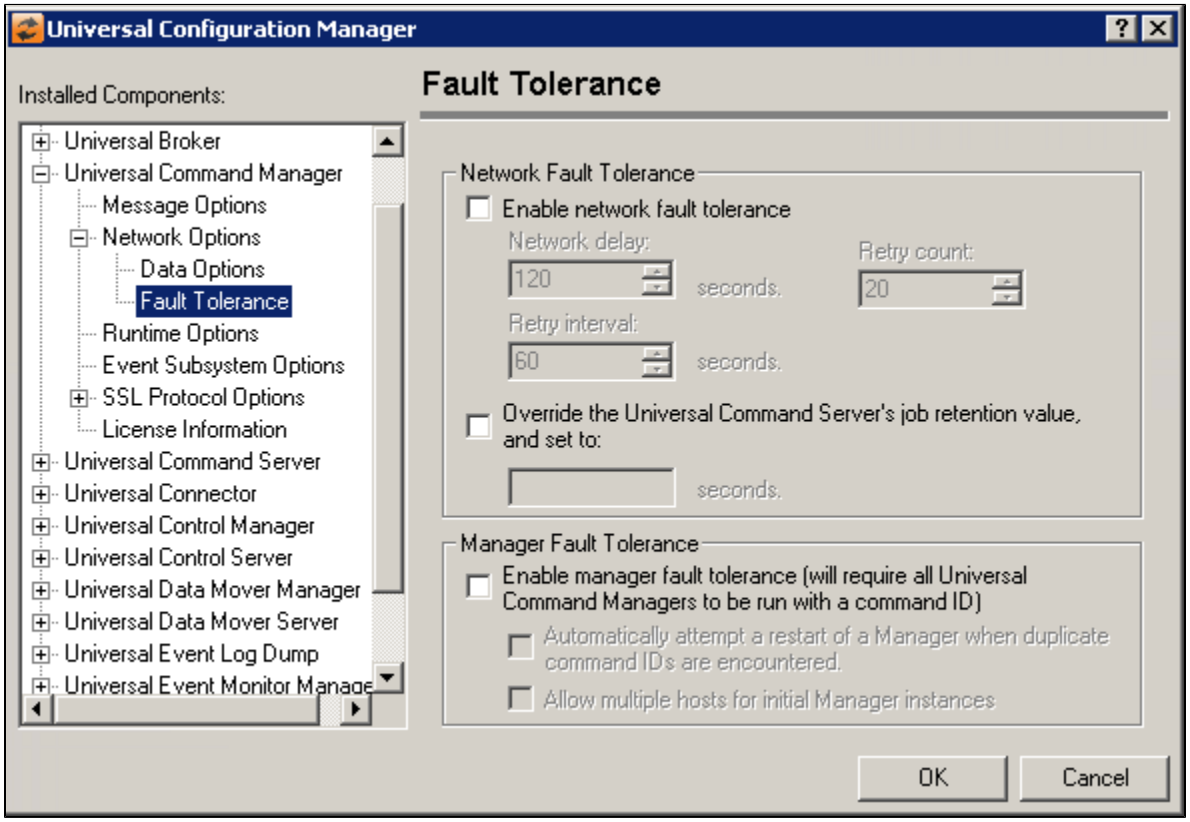
t.

Universal Command Manager: Network Options - Data Options



Field	Description
Network Data Transfer Options	**Compress Data** Controls compression of data transmitted across the network. Select the checkbox to turn this option on. **Enable maximum compression** Select this option to use the zlib compression algorithm, which usually results in a very high compression rate. When this option is not selected, the hasp compression algorithm will be used. This algorithm is not as CPU-intensive as the zlib algorithm, but the resulting data is a little less compressed. **Encrypt Data** Controls encryption of data transmitted across the network. Select the checkbox to turn this option on. **Verify Data Integrity** Controls whether or not data integrity is verified. When this option is selected, authentication checks will be done to insure that the data sent is the same data received. If the data changed, the data integrity is jeopardized and the transmission terminates.
Code Page	Sets the translation table used to translate text-based data transmitted across the network.

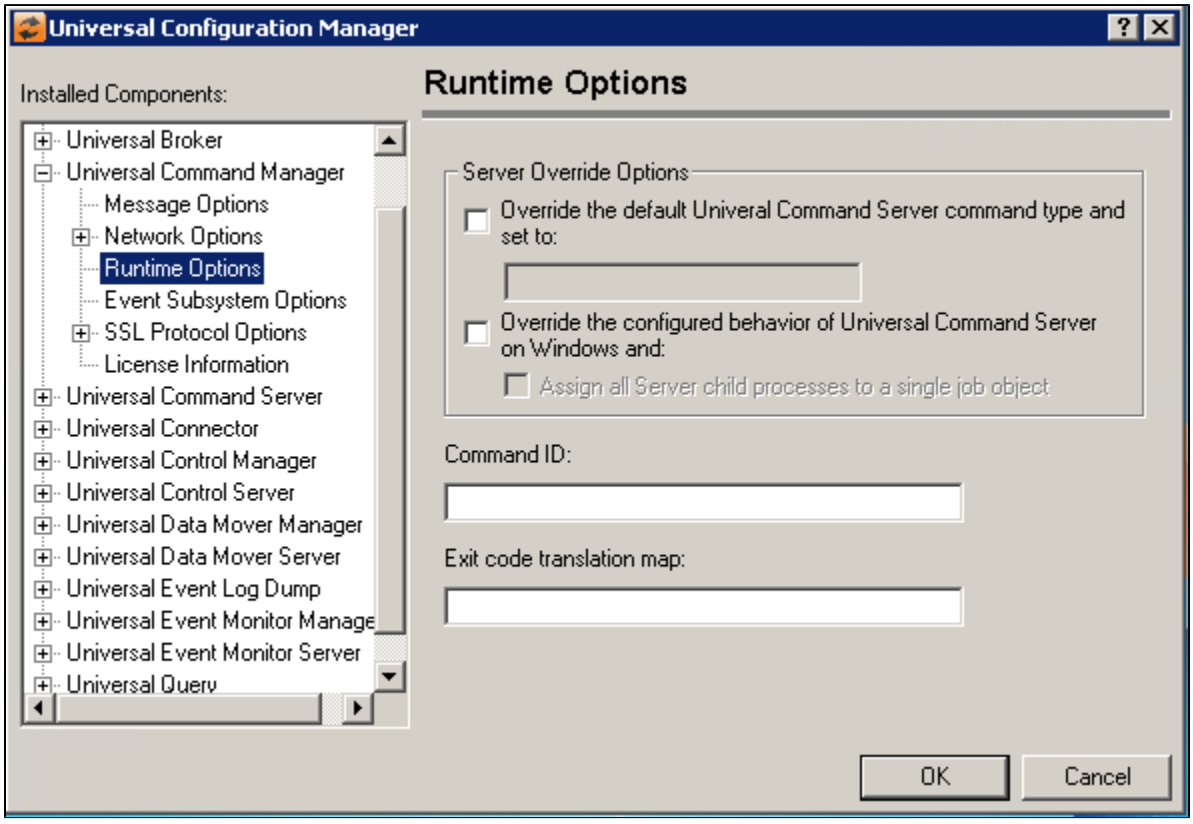
Universal Command Manager: Network Options - Fault Tolerance



Field	Description
Network Fault Tolerance	**Enable network fault tolerance** Enables the network fault tolerance feature. **Network Delay** The number of seconds (from 1 to 900) of network inactivity before the need for a reconnect is assumed. **Retry Count** The maximum number of reconnects attempted (between 1 and 900). **Retry interval** The number of seconds (between 0 and 900) between each reconnect attempt. **Override the Universal Command Server's job retention value** The job retention option specifies the time to retain the exit status of the command executed after termination of the command if the

	network connection to the manager is down. This allows a manager to reestablish the connection after the command has completed. Selecting this option will override the value in the Universal Command Server's configuration.
Manager Fault Tolerance	**Enable Manager Fault Tolerance** When this option is selected, manager fault tolerance will be requested for each instance of a Universal Command Manager that is run from this location. This means that if the remote Universal Command Server supports manager fault tolerance, and is allowing redirected i/o to be spooled, the Universal Command Manager may be restarted. Please note that when this option is selected, any Universal Command Manager run from this location will require a command ID (specified with the -C or -cmdid command line option). **Automatically attempt a restart of a Manager when a duplicate command IDs are encountered** When this option is selected, the restart configuration option is set to "auto". This means that when an instance of Universal Command Manager is started at this location, it will first try to start a new instance of a Universal Command Server. If a Server with a matching command ID is found, instead of failing, the Manager will automatically attempt a restart. Selecting this option is useful when Universal Command is run via scripts that are not necessarily easy to change when a restart is required. When this option is not selected, the restart configuration option is set to "no". This means that if a Universal Command Manager is started with a command ID that is already in use (that is, it belongs to an active Universal Command Server component), the Manager will fail. Before this option can be selected, manager fault tolerance must be enabled. This means that any Universal Command Manager started at this location must have a command ID, specified with the -C or -cmdid command line option. **Allow multiple hosts for initial Manager interfaces** When this option is selected and manager fault tolerance is enabled, multiple hosts in the host list are allowed only when a Universal Command Manager is submitted for the first time (that is, its restart option is set to no). Care must be taken when setting this option, because a situation could result in which multiple Universal Command Servers are started on different hosts for the same Universal Command Manager. For example, if a Manager was terminated because of a failed network connection, and that network connection is still down when that cancelled Manager is resubmitted, there is no way to know that the Server on the unreachable system is awaiting restart. This means a a new Server could be started on any host in the list that the Manager can connect to. Multiple hosts in the host list are never permitted when fault tolerant Managers are configured to automatically restart, or when a Manager is explicitly instructed to restart (that is, the restart option is set to auto or yes, respectively). In this case, the host to which the Manager originally connected must be provided as a single entry in the host list. This option is ignored if manager fault tolerance is not enabled. In that situation, multiple hosts are always permitted. This option is not selected by default.

Universal Command Manager: Runtime Options



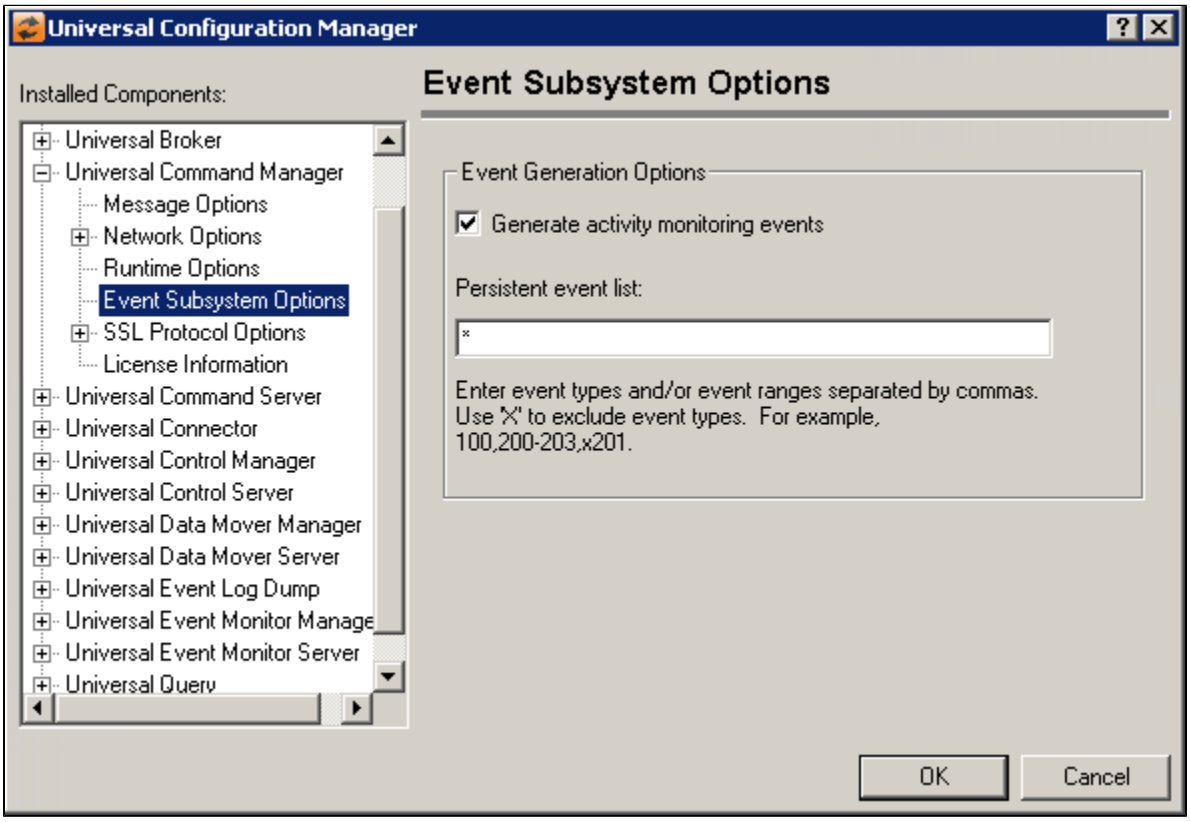
Field	Description

Server Override Options	**Override the default Universal Command Server command type and set to** When this option is checked, the Univeral Command Manager will send the specified command type to a Universal Command Server, overriding the Server's configured value. Leave this option unchecked to cause the remote Universal Command Server to use its configured default. The command type specified here will depend on the platform and version of the remote Universal Command Server. Not all command types are available on all platforms. Also, the version of the remote Universal Command Server may not offer support for command types, or may only provide support for a subset of command types recognized by the installed version of Universal Command Manager. The command types recognized by this version of Universal Command Manager for Windows includes (but is not limited to): • cmdref Command reference • shell Shell • stc Started task If the default command type you wish to use is not shown in this, you may still enter it here. The value will not be validated until it is sent to the Universal Command Server. See the Universal Command Consolidated Documentation for a complete list of command types supported by a Universal Command Server on a particular platform. **Override the configured behavior of the Universal Command Server on windows and:** Check this option to set a flag within the Universal Command Manager's configuration file that overrides how the relationship among user processes is managed for a specified Universal Command Server on Windows. The Universal Command Server normally defines this relationship, but enabling this option allows the local Universal Command Manager to override the Server's configured default. Leave this option unchecked to remove the value from the local Universal Command Manager's configuration, which causes the specified Universal Command Server to use its configured default. Note: Checking this option does not set the configuration value itself; it only controls whether or not the Universal Configuration Manager sets the override in the Univeral Comand Manager's configuration file. **Assign all Server child processes to a single job object** Check this option to instruct a specified Windows Universal Command Server to assign the child process it creates – along with any descendants – to a single entity known as a job. Assigning a process to a job simplifies process management on Windows, by causing all child processes (that is, the requested user process and any of its child processes) to terminate whenever their parent is cancelled (that is, Universal Command Server). When this option is unchecked, cancelling a Universal Command Server on Windows will only result in the termination of its immediate child processes. Any processes that child spawns will continue to run after the Server component is cancelled. Note: Setting this option in the Universal Command Manager's configuration file overrides the configured value for any Windows Universal Command Server specified by the local Manager. This option is only recognized by Universal Command Servers 3.2.0.0 and later, running on Windows. It is ignored for all other platforms and versions.
Command ID	The command ID helps associate a Universal Command Server component with the Universal Command Manager that started it. Any value may be specified for this option, but a UCMD Manager will fail when Manager fault tolerance is enabled, if another UCMD Manager with a matching command ID is active or awaiting restart. Set this option to an asterisk (*) to instruct UCMD to automatically generate a unique command ID each time it runs.
Exit code translation map	The exit_code_map option is used to translate the exit code issued by a user process (executed by UCMD Server) to a UCMD Manager exit code. The exit code map has the following format: `[exitStatus]range:exitCode[,...[exitStatusN]rangeN:exitCodeN]` **exitStatus:** optional, may be 'a' or 'n'. Used to indicate whether the mapping applies to processes that end (a)bnormally or (n)ormally. If this parameter is not specified, the process's termination status is ignored. **Range:** The range of exit codes to be mapped. 'range' may be a single value, an series of values separated by a hyphen (for example, 1-5), or an upper or lower limit of values, preceded by the less than ('<') or the greater than ('>') symbol, respectively (for example, <20 or >20). An asterisk ('*') can be used to indicate all process exit codes. **exitCode:** The value UCMD Manager will return when the user process ends with a matching exit status and exit code. Examples: • n*:1,a*:16 Map all normal terminations to exit code 1 and all abnormal terminations to exit code 16. • 1-5:1,>5:2 Map exit codes 1 thru 5 (inclusive) to exit code 1 and exit codes greater than 5 to exit code 2.

- <0:64,>1:1
  Map exit codes less than 0 to exit code 64 and exit codes greater than 1 to 1

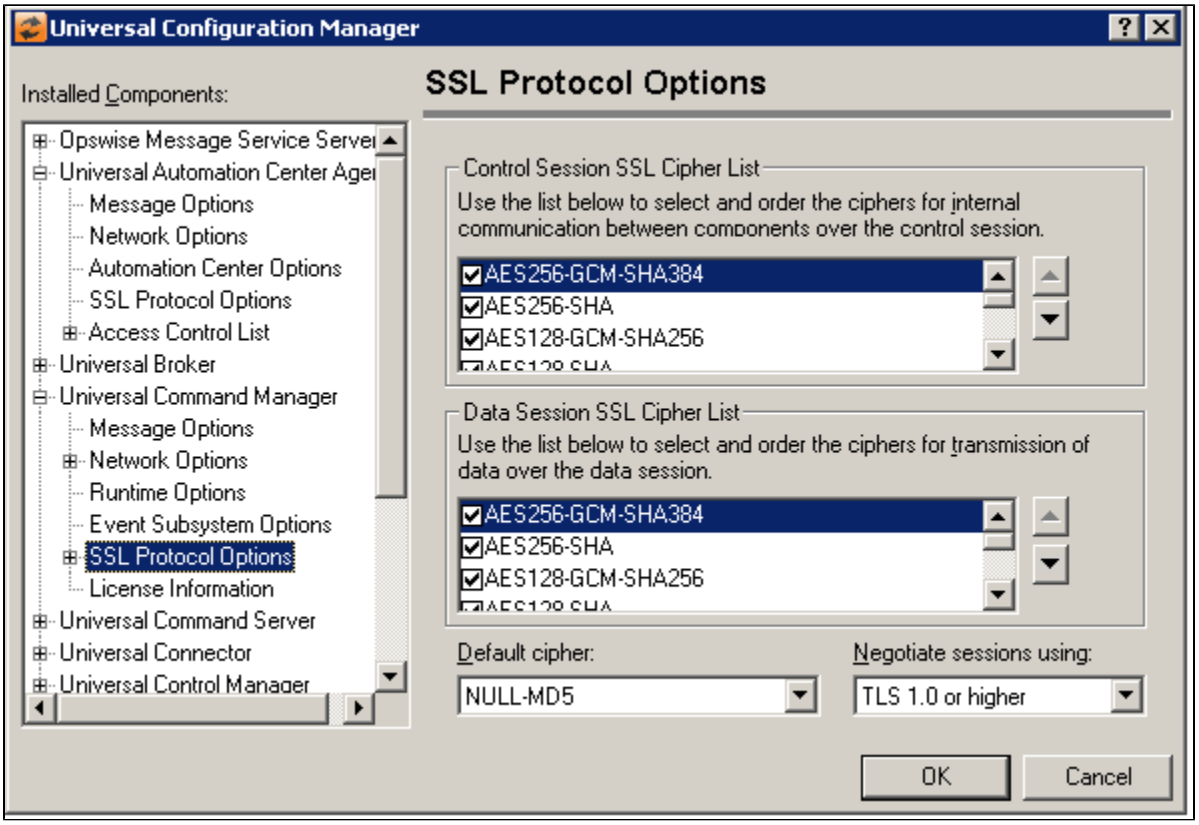
If the exit code from a user process matches more than one entry in the list, the first matching entry is applied.

Universal Command Manager: Event Subsystem Options

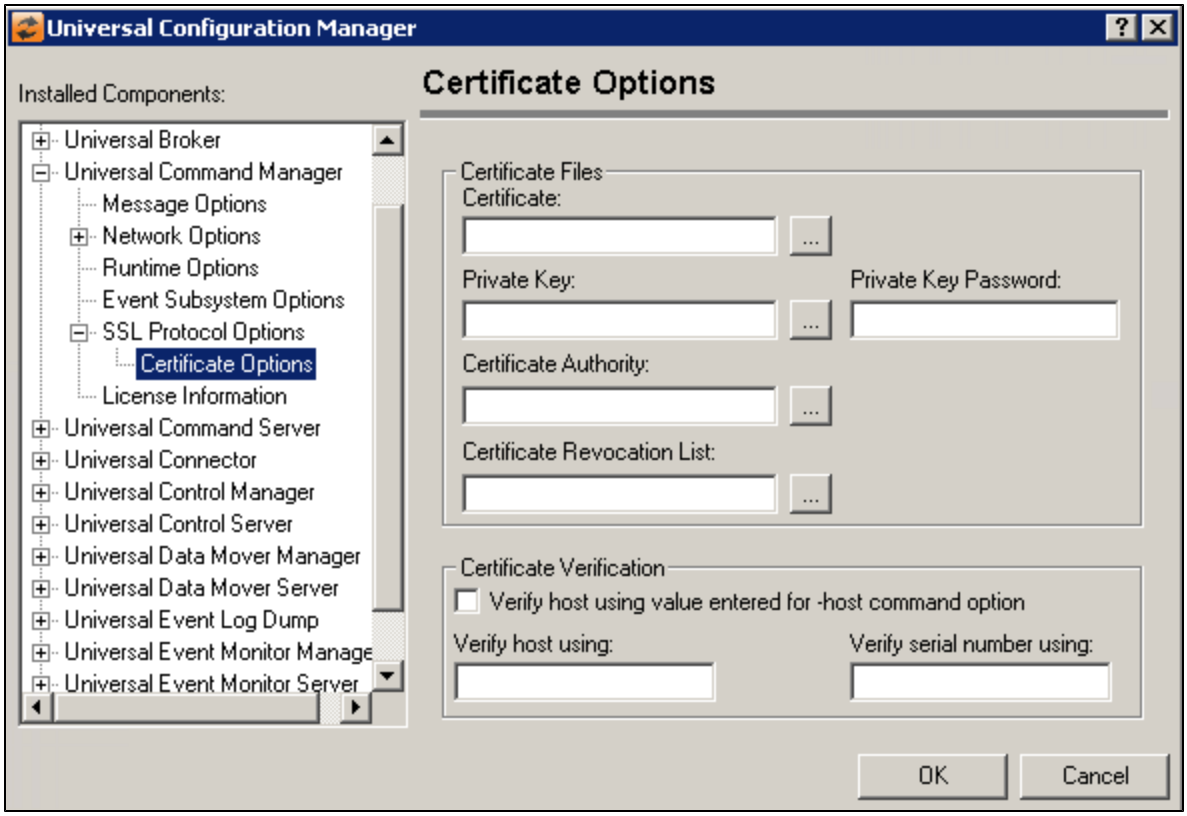


Field	Description
Event Generation Options	**Generate activity monitoring events** When this option is selected, the Universal Command Manager will generate events that monitor product activity. This option is selected by default. **Persistent event list** Specifies which events are to be generated and processed as persistent events. A persistent event is saved in a Universal Enterprise Controller (UEC) event database for long-term storage. Enter a list and/or range of numeric event types, separated by a comma. To exclude an event or range of events, precede the entry with an upper- or lower-case 'X'. Use an asterisk to specify all event types. For example: • x* Do not generate any events (this is the default) • * Generate all event types • 100 Generate all event types except 100 • 100,200-205 Generate event type 100 and 200 through 205 • 100,200-205,x202 Generate event types 100 and 200 thru 205, excluding event type 202 Entries are processed from left to right, and the list is order dependent. For example, an entry of x*,100 prevents generation of all event types except 100, while an entry of 100,x* prevents generation of all event types (the right-most entry takes precedence).

Universal Command Manager: SSL/TLS Protocol Options

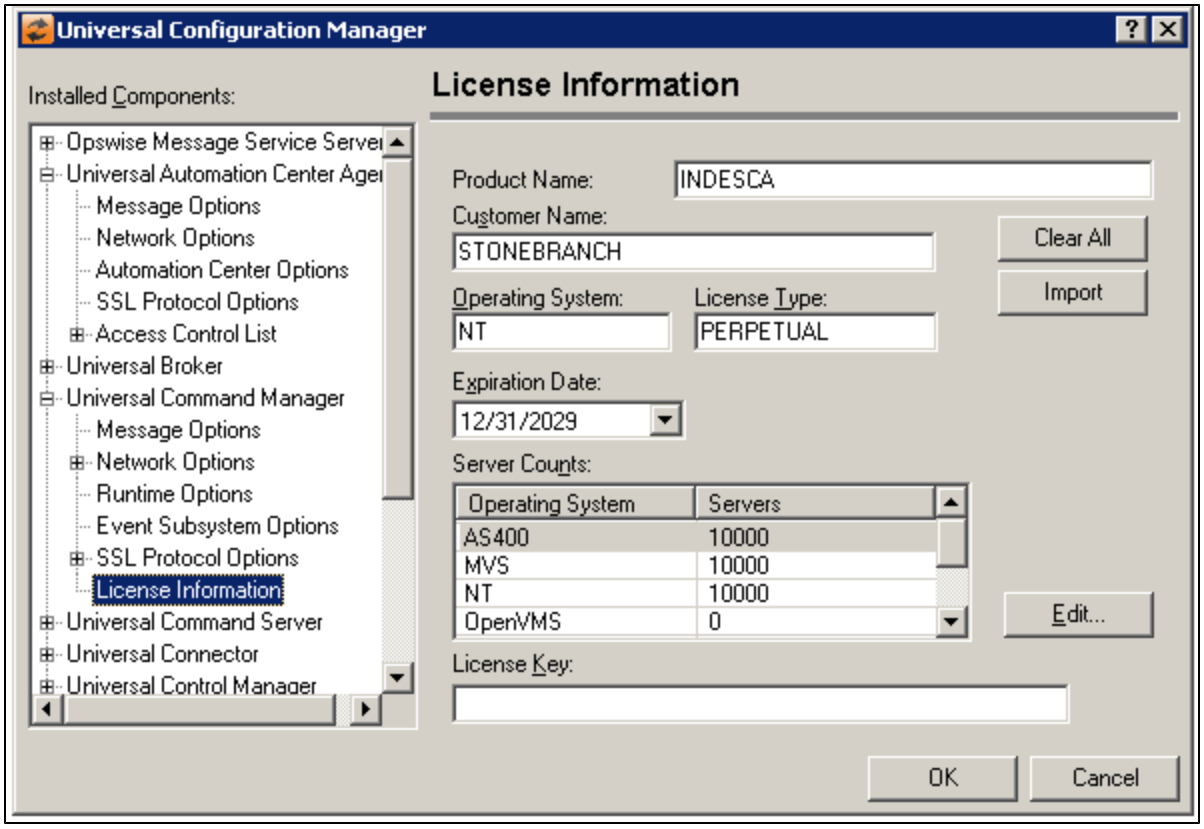


Field	Description
Control Session SSL Cipher List	The Control Cipher List specifies one or more acceptable cipher suites to use for network communication. The list consists of one or more cipher suites ordered with the most preferred suite first and the least preferred suite last. To disable SSL/TLS encryption and message authentication for messages transmitted across the control session, select the NULL-NULL entry. Use of this cipher requires that the remote UCMD Server's encrypt_control_session option be set to no. Otherwise, the UCMD Server will issue a protocol error when the Manager tries to establish the control session with it.
Data Session SSL Cipher List	The data ssl cipher specifies the cipher suites for the standar I/O connections between the UCMD components when encryption is specified.
Default cipher	(No help provided.)
Negotiate sessions using	(No help provided.)



Field	Description
Certificate Files	**Certificate** The Certificate option specifies the X.509 certificate file that identifies the program to remote components with which it connects using the SSL/TLS protocol. **Private Key** The Private key option specifies the private key file associated with the X.509 certificate specified in the Certificate file. **Private Key Password** The Private key password specifies the password associated with the private key file. **Certificate Authority** The Certificate Authority option specifies a file with a list of trusted authorities. Any certificate received that is signed by a trusted Certificate Authority is accepted. **Certificate Revocation List** The Certificate Revocation List option specifies a file with a list of revoked certificates. Any certificate received that is in this list will not be accepted.
Certificate Verification	**Verify host using value entered for -host command option** The host name specifies that the ubroker certificate's host name value must match either the -host option value or the value specified by the entered host name. There is no default. **Verify host using** The host name specifies that the ubroker certificate's host name value must match either the -host option value or the value specified by the entered host name. There is no default. **Verify serial number using** The serial number specifies that the ubroker's X.509 certificate serial number is verified to match the value specified. The number is considered to be in decimal format unless it is prefixed with 0x or 0X in which case it is considered a hexadecimal format. There is no default.

Universal Command Manager: License Information



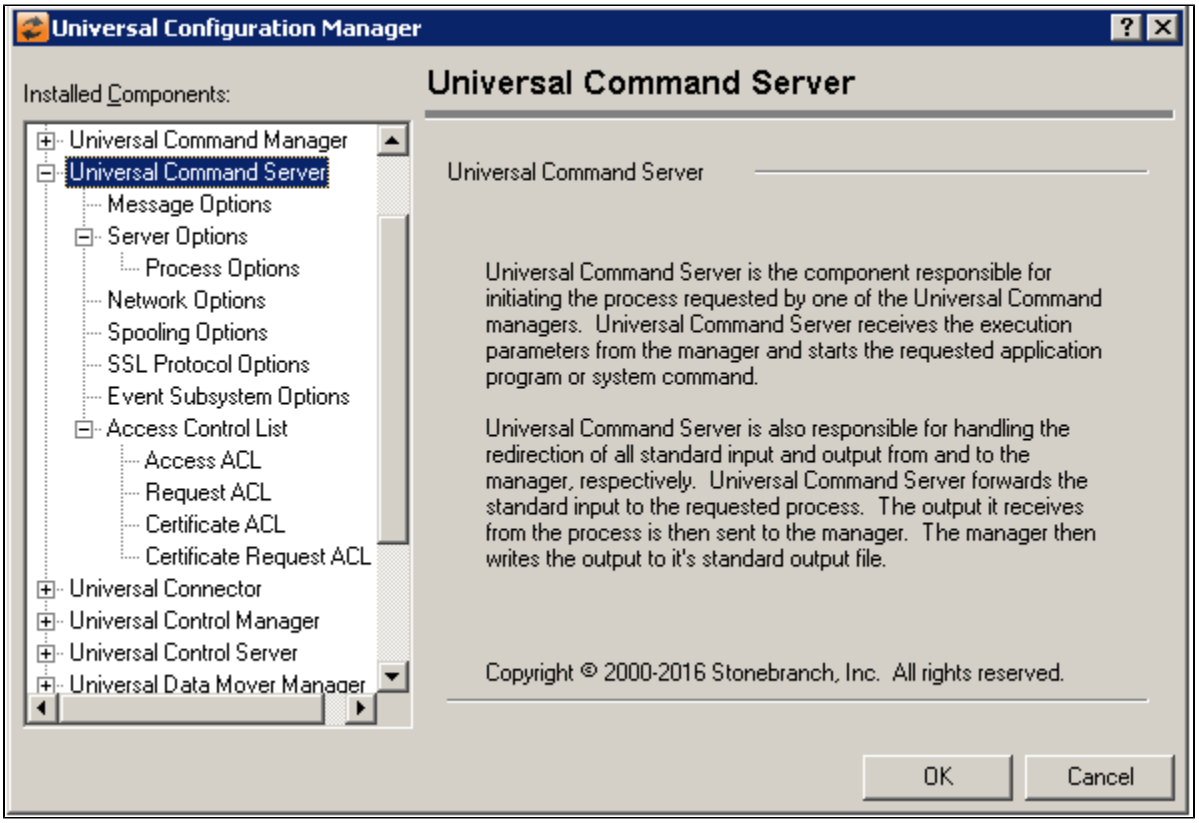
Field	Description
Product Name	Specifies the name of the registered product for which the license key was issued.
Customer Name	Contains the name under which the software is registered. This value must be entered exactly as the customer name specified in the licensing material you received with the software.
Operating System	Identifies the operating system on which your licensed Universal Command Manager will run. This value should match the operating system specified in the licensing material you received with your software.
License Type	Specifies the type of license issued for the current product. This value is used with the expiration date to determine the length of time the software license is valid.
Expiration Date	Contains the date when the current license for this product will expire. The date entered must match the expiration date specified in the licensing material you received with the software. To change the date, enter a new value, or click the down arrow to display a calendar control.
Server Counts	Contains the number of Universal Command Servers you have licensed, by operating system. This values in this list should match the numbers specified in the licensing material you received with your software.
License Key	Contains the license key that is generated for each product, customer, license type and expiration date. The value contained in this field must match the key specified in the licensing material you received with the software package.
Clear All Button	(No help provided.)
Edit Button	Displays a dialog from which the number of licensed servers for the selected operating system can be updated.
Import Button	Select this button to browse for and load the Universal Command Manager license file, which will automatically import the license information.

Universal Configuration Manager - Universal Command Server

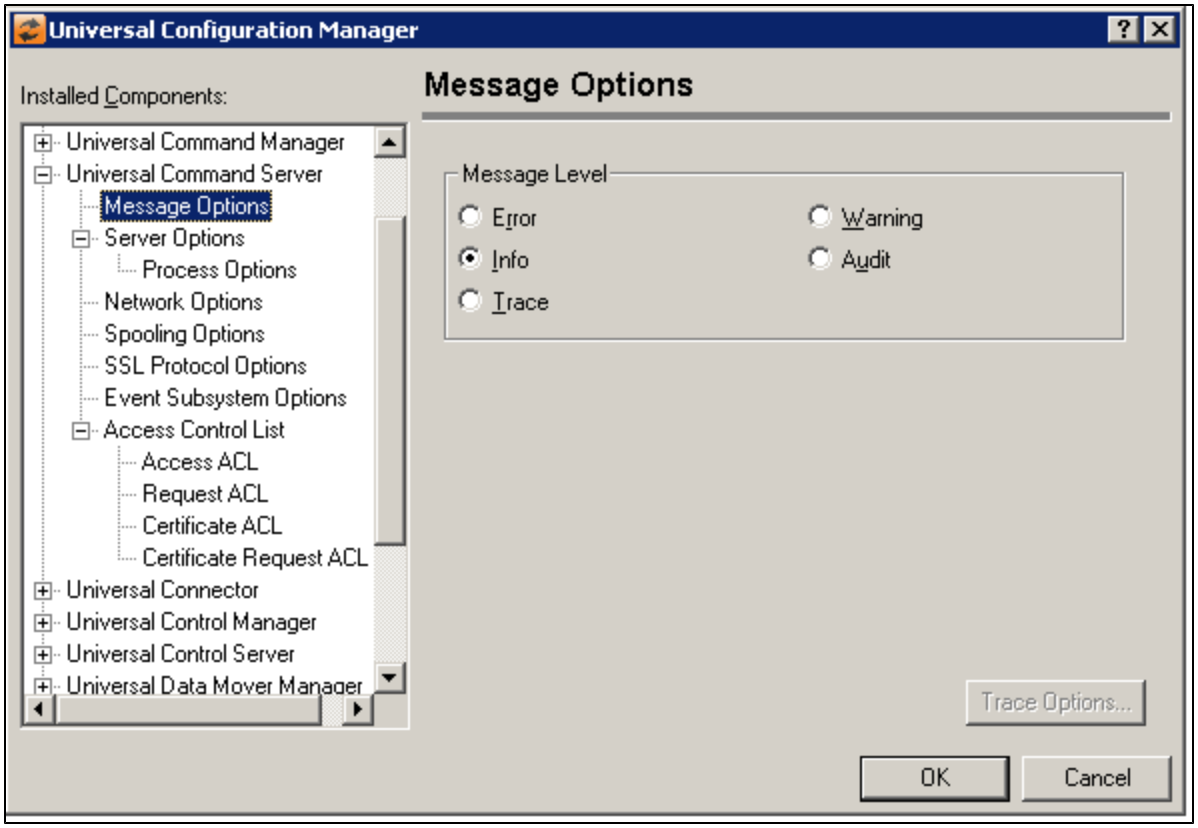
- Universal Command Server

- Universal Command Server: Message Options
- Universal Command Server: Server Options
  - Universal Command Server: Server Options - Process Options
- Universal Command Server: Network Options
- Universal Command Server: Spooling Options
- Universal Command Server: SSL/TLS Protocol Options
- Universal Command Server: Event Subsystem Options
- Universal Command Server: Access Control List
  - Universal Command Server: Access Control List - Access ACL
  - Universal Command Server: Access Control List: Request ACL
  - Universal Command Server: Access Control List - Certificate ACL
  - Universal Command Server: Access Control List: Certificate Request ACL

Universal Command Server

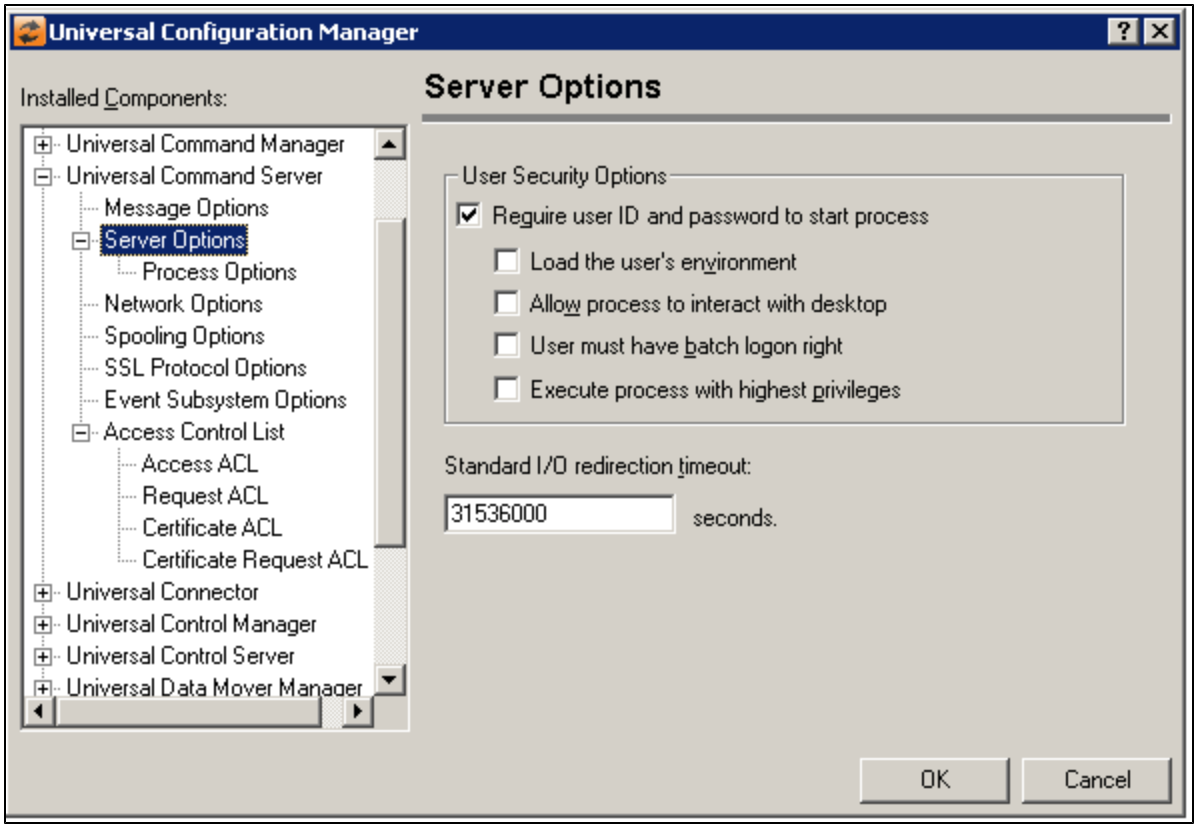


Universal Command Server: Message Options

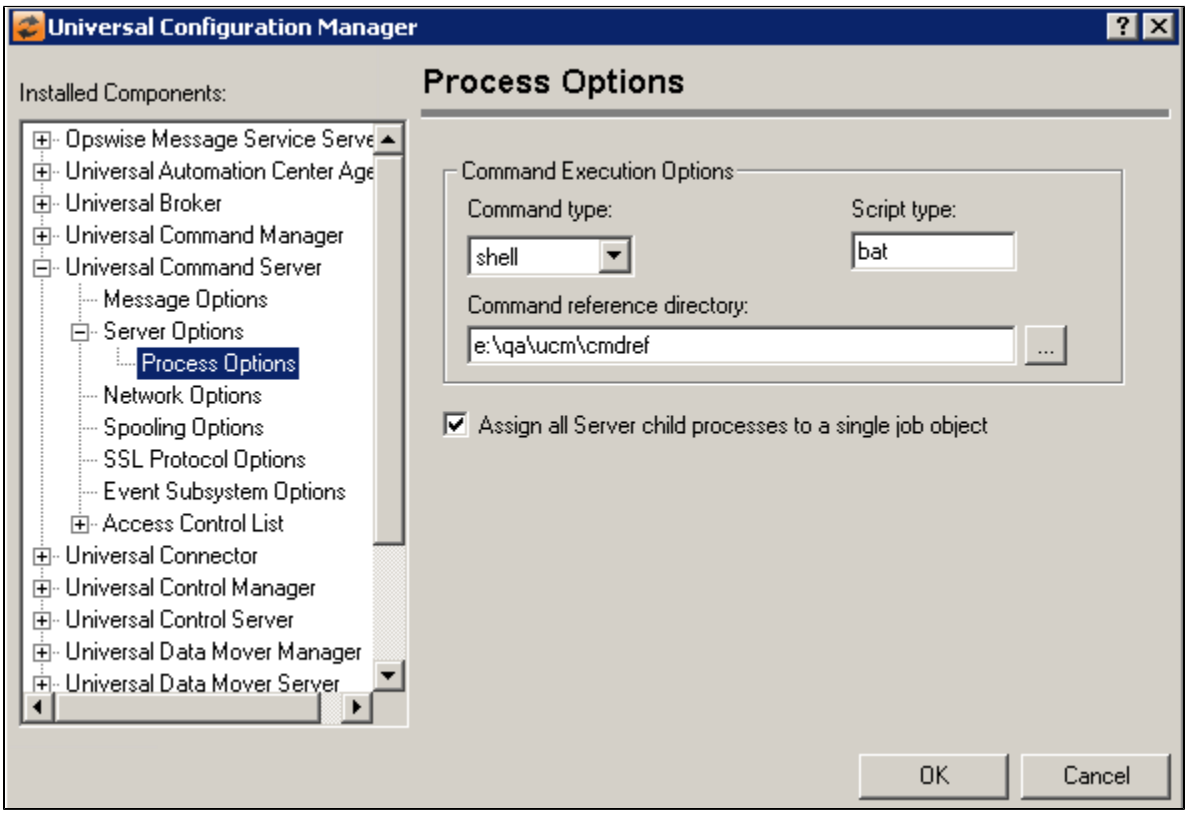


Field	Description
Message Level	**Error** Turns on message logging for errors only. **Warning** Turns on message logging for warnings and errors. **Info** Turns on message logging for all informational, warning and error messages. **Audit** Turns on message logging for all audit, informational, warning and error messages. **Trace** Allows all messages to be logged, including those used for debugging purposes.

Universal Command Server: Server Options

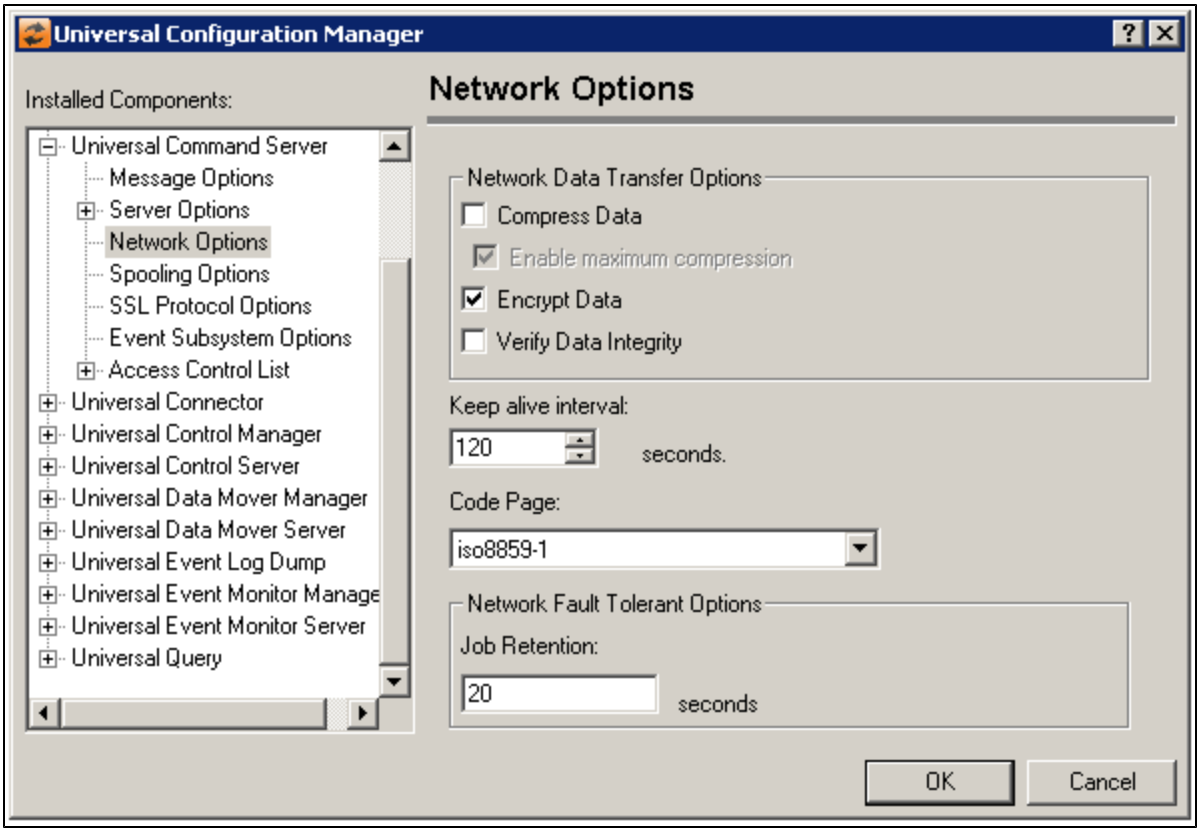


Field	Description
User Security Options	**Require User ID and password to start process** When this option is checked, Universal Command will require a valid user ID and password to start a process. **Load the user's environment** When checked, causes Universal Command Server to load the profile and the environment settings for the user account being used to execute the requested process. This is necessary only if the process requires access to user-specific environment variables, or registry values that are only stored under the HKEY_CURRENT_USER key. **Allow process to interact with desktop** When this option is checked, processes are run in a context that permits the current console logon session to interact with them. This also allows the process to invoke system functions that access desktop elements (for example, Windows, menus, buttons) associated with that session. Accordingly, this exposes a risk whereby a process could hijack the desktop and invoke malicious code using a (potentially) elevated security context. • Leaving this option unchecked causes user processes to execute in a context that isolates them from the current interactive logon session. Unless processes that require user interaction are executed, this is the recommended setting. • This option only applies if a user account (in whose security context a process is executed) is authenticated via an interactive logon. If the User must have batch logon right option is checked, a batch logon is done to establish the process's security context. This logon method requires additional privileges, but disallows all interaction with the desktop. **User must have batch logon rights** When this option is checked, the Log on as a batch job user right must be granted to the user ID passed to Universal Command before the process will run. This is an advanced user right that must be set using the NT User Manager. **Execute process with highest privileges** When this option is checked, user processes that UCMD Server executes receive an elevated access token. This gives the account used to execute the process the highest privileges available to it. Note that the account is not granted any new privileges, it simply receives the privileges necessary to execute tasks under User Account Control (UAC) restrictions. Note that this option is only recognized for Windows platforms starting with Windows Vista. It is ignored for Windows XP and Server 2003.
Standard I/O redirection timeout	Specifies the length of time, in seconds, that a Universal Command Server will wait for stdin, stdout and stderr files of the process it starts to close. If the closure of those files is not detected within the time limit set by this value, the Universal Command Server will end, even if the process it started has not ended.



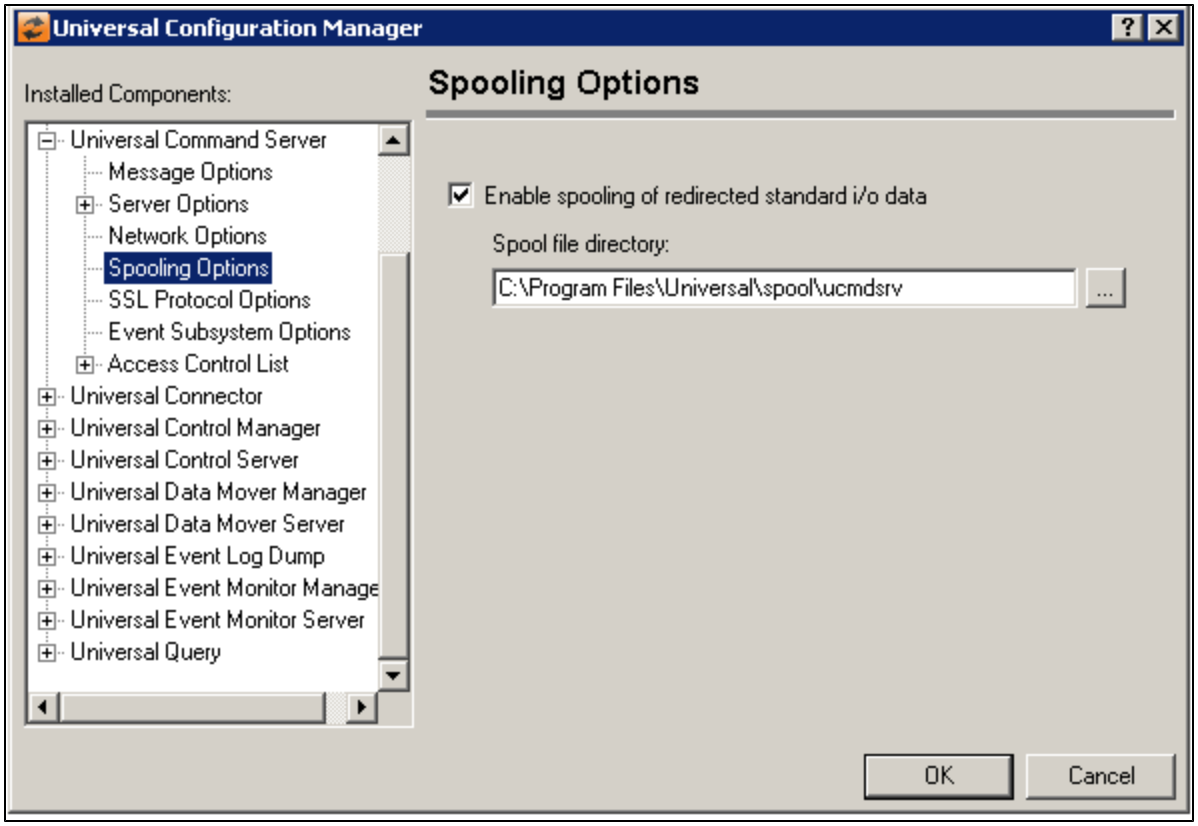
Field	Description
Command Execution Options	**Command Type** The command type option specifies the default command type when one is not specified directly by the manager. The default is shell. **Script Type** The value type specifies the default script type when a Manager does not specify one. Windows script types are interpreted as file extensions. The script is executed by the program associated with the file extension. If no program association exists for the extension, the script will fail. The default is BAT. **Command reference directory** The command reference directory specifies where Universal Command Server searches for command references.
Assign all Server child processes to a single job object	This option instructs a Universal Command Server how to manage the relationship between it and the child processes it creates. When this option is checked, the child process Universal Command Server starts – and any of that process's descendants – are assigned to a single entity known as a job. Assigning a process to a job simplifies process management, by causing all child processes (that is, the requested user process and any of its child processes) to terminate whenever their parent is cancelled (that is, Universal Command Server). When this option is NOT checked, no relationship among child processes is maintained, and each child process must be terminated individually when a Universal Command Server component is cancelled. This is the behavior exhibited in Universal Command Server releases prior to v3.2.0.0. Note: The value for this option may be overridden from the remote Universal Command Manager's command options on a per-request basis.

Universal Command Server: Network Options



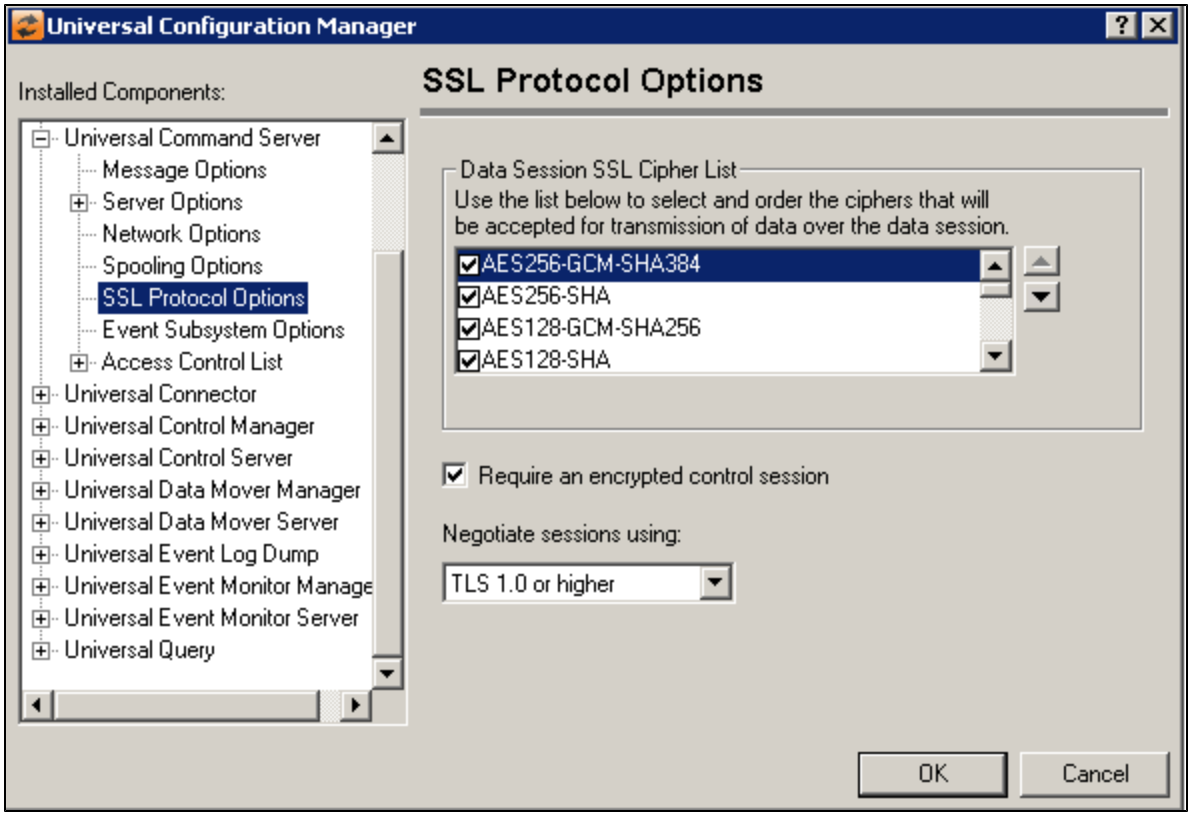
Field	Description
Network Data Transfer Options	**Compress Data** Controls compression of data transmitted across the network. Select the checkbox to turn this option on. **Enable max compression** Select this option to use the zlib compression algorithm, which usually results in a very high compression rate. When this option is not selected, the hasp compression algorithm will be used. This algorithm is not as CPU-intensive as the zlib algorithm, but the resulting data is a little less compressed. **Encrypt Data** Controls encryption of data transmitted across the network. Select the checkbox to turn this option on. **Verify Data integrity** Controls whether or not data integrity is verified. When this option is selected, authentication checks will be done to insure that the data sent is the same data received. If the data changed, the data integrity is jeopardized and the transmission terminates.
Keep live Interval	This value specifies the frequency with which a keep alive message will be sent to the Universal Command Manager. The manager uses this message to verify that a network connection still exists with the server during periods of network inactivity.
Code Page	Sets the translation table used to translate text-based data transmitted across the network.
Network Fault Tolerant Options	**Job Retention** The job retention option specifies the time to retain the exit status of the command executed after termination of the command if the network connection to the manager is down. This allows a manager to reestablish the connection after the command has completed. The value is stored in seconds, and the default is 172800 seconds, or 2 days.

Universal Command Server: Spooling Options



Field	Description
Enable spooling of redirected standard i/o data	Controls whether or not the Universal Command Server will allow redirected input and output to be spooled. When checked, the Universal Command Server will spool redirected i/o, provided the Universal Command Manager has requested it. If this option is not checked, no spooling will be done, and all requests by Universal Command Managers to do so will be rejected.
Spool file directory	The directory where spool files for each component instance will be stored. This directory must be located on local storage. It may not be located on any storage area accessed via the network which includes but is not limited to NFS, SAMBA, Microsoft File Sharing.

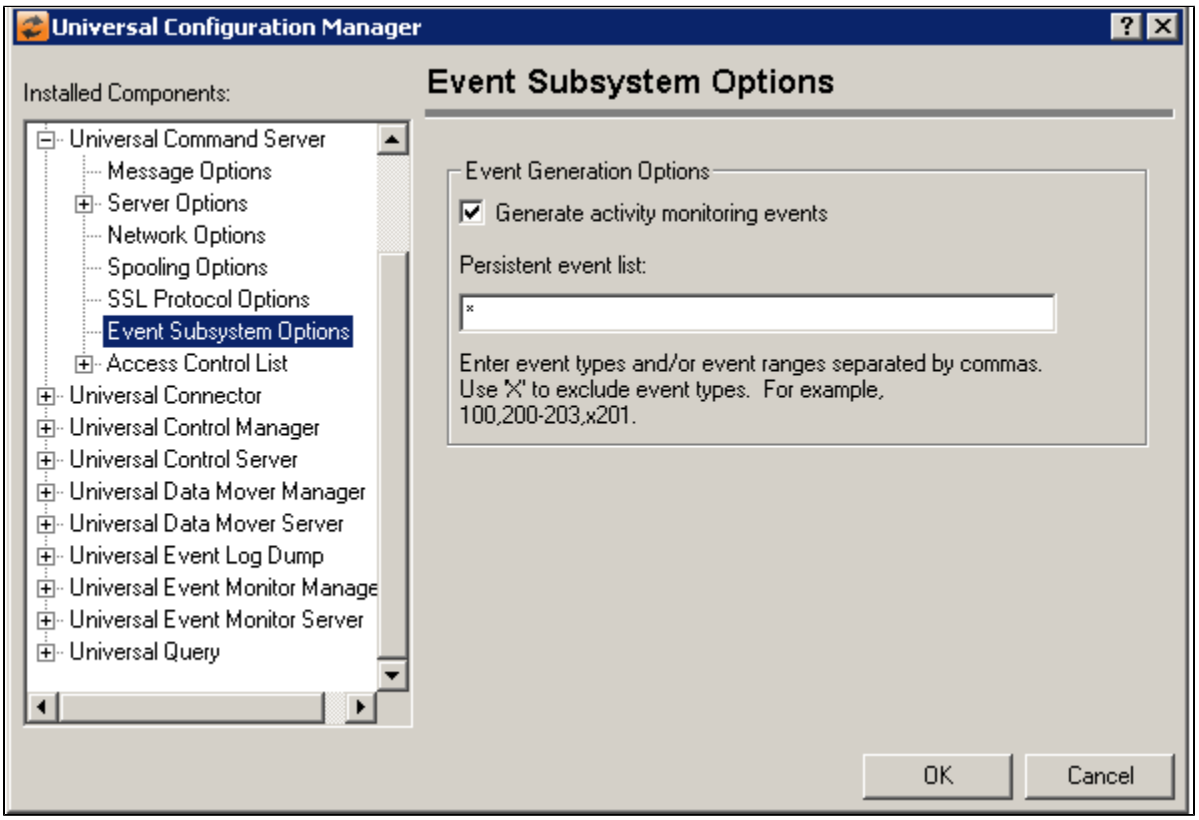
Universal Command Server: SSL/TLS Protocol Options



Field	Description
Data Session SSL Cipher List	Universal Command uses the SSL/TLS protocol for data encryption. SSL/TLS has a variety of encryption and message digest algorithms that can be used for network communication. The SSL/TLS cipher suites available for network communication over the data session are listed here. The cipher suites to use may be specified by checking the box to the left of the cipher name. The order in which the suites are applied may also be specified by selecting the up/down arrows to the right of the list. Please note that the order may be changed for any of the ciphers listed, but only those with a check next to their name will be used. The following cipher suites are available: • RC4-SHA 128-bit RC4 encryption, SHA-1 message digest • RC4-MD5 128-bit RC4 encryption, MD5 message digest • AES256-SHA 256-bit AES encryption, SHA-1 message digest • AES128-SHA 128-bit AES encryption, SHA-1 message digest • DES-CBC3-SHA 128-bit Triple-DES encryption, SHA-1 message digest • DES-CBC-SHA 128-bit DES encryption, SHA-1 message digest • NULL-SHA No encryption, SHA-1 message digest • NULL-MD5 No encryption, MD5 message digest
Require and encrypted control session	Select this option to require encryption over the control session between the UCMD Manager and Server. When this option is selected, the UCMD Server will not accept a control session SSL/TLS cipher of NULL-NULL from the remote UCMD Manager. When this option is not selected, the UCMD Server will accept any control session cipher from the UCMD Manager, even one that disables SSL/TLS encryption. This option is checked by default.
	(No help available.)

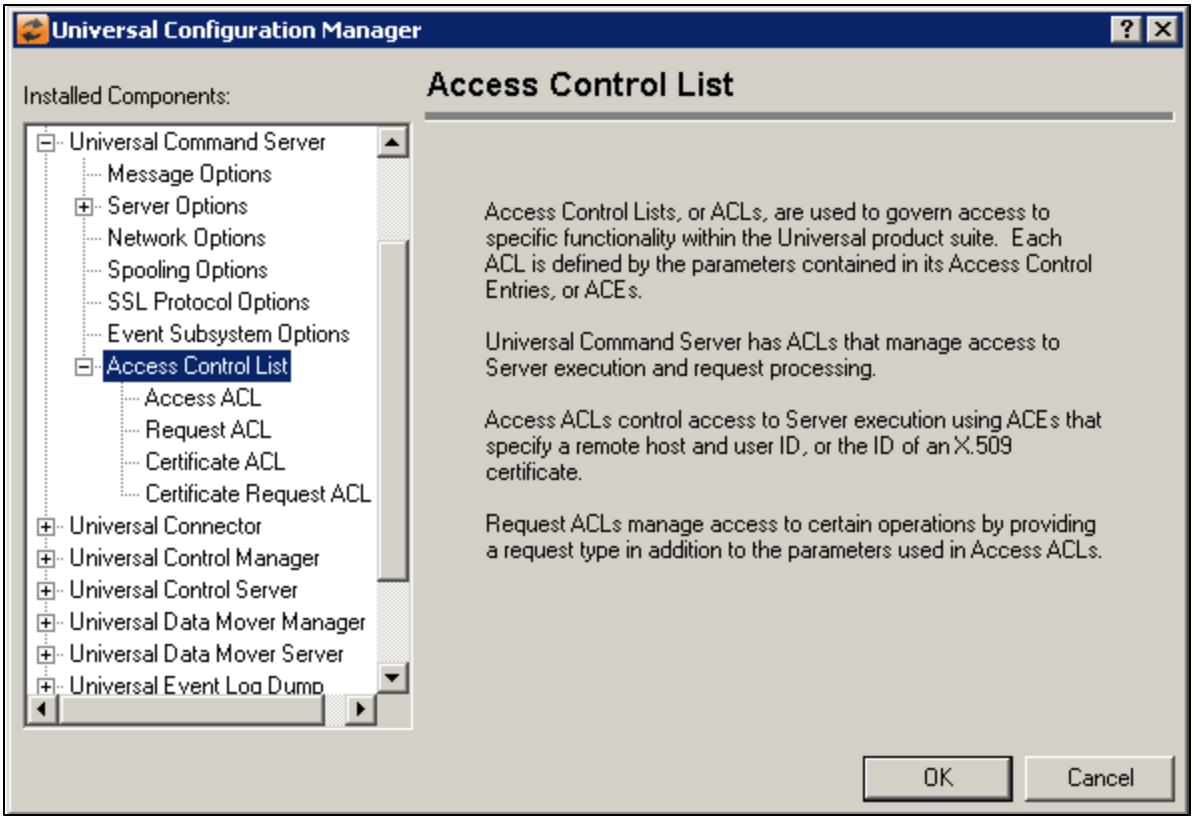
Negotiate sessions using	

Universal Command Server: Event Subsystem Options

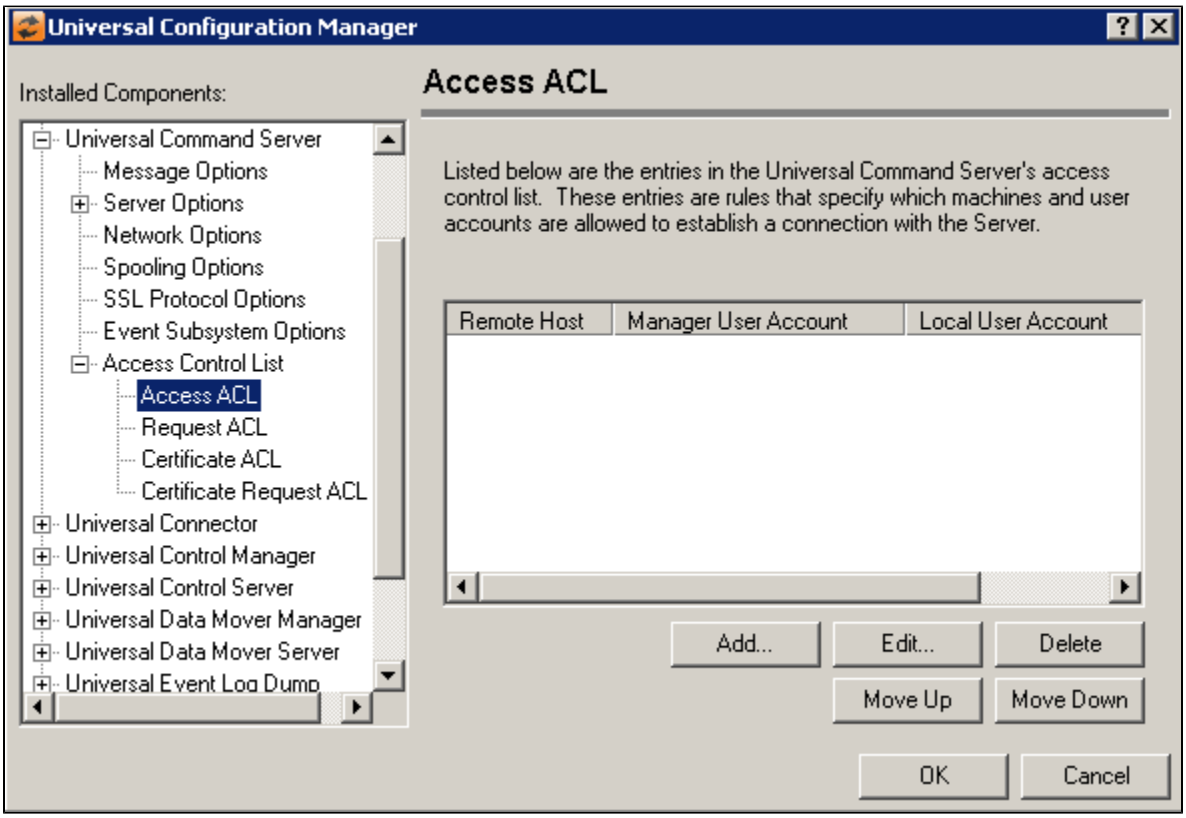


Field	Description
Event Generation Options	**Generate activity monitoring events** When this option is selected, the Universal Command Server will generate events that monitor product activity. **Persistent event list** Specifies which events are to be generated and processed as persistent events. A persistent event is saved in a Universal Enterprise Controller (UEC) event database for long-term storage. Enter a list and/or range of numeric event types, separated by a comma. To exclude an event or range of events, precede the entry with an upper- or lower-case 'X'. Use an asterisk to specify all event types. For example: • x* Do not generate any events (this is the default) • * Generate all event types • 100 Generate all event types except 100 • 100,200-205 Generate event type 100 and 200 through 205 • 100,200-205,x202 Generate event types 100 and 200 thru 205, excluding event type 202 Entries are processed from left to right, and the list is order dependent. For example, an entry of x*,100 prevents generation of all event types except 100, while an entry of 100,x* prevents generation of all event types (the right-most entry takes precedence).

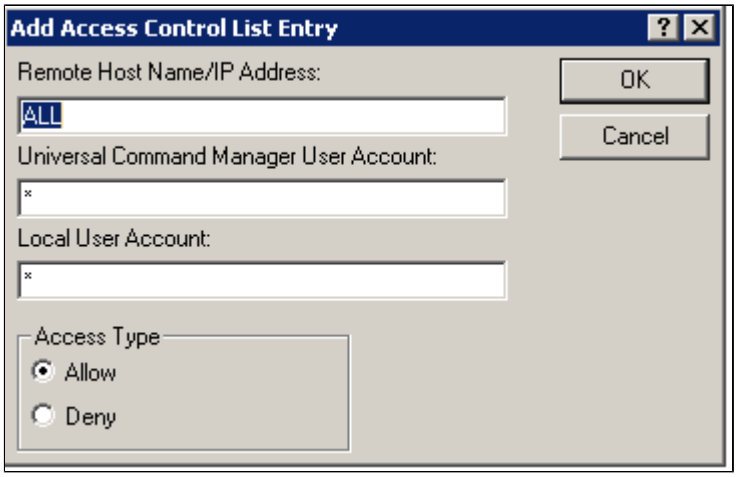
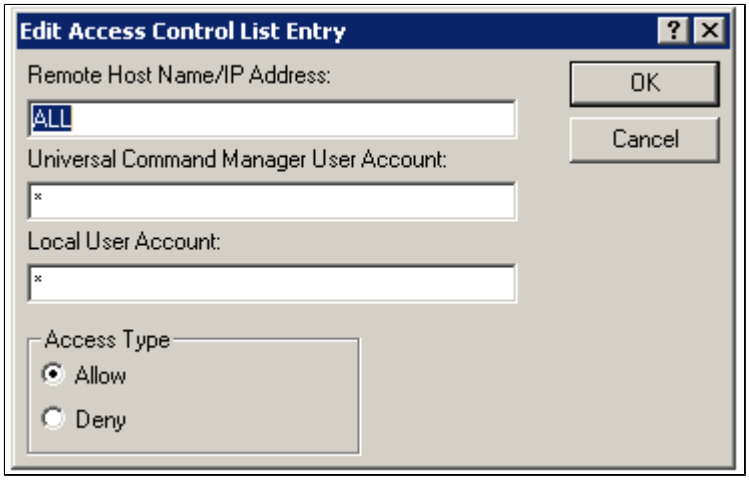
Universal Command Server: Access Control List



Universal Command Server: Access Control List - Access ACL

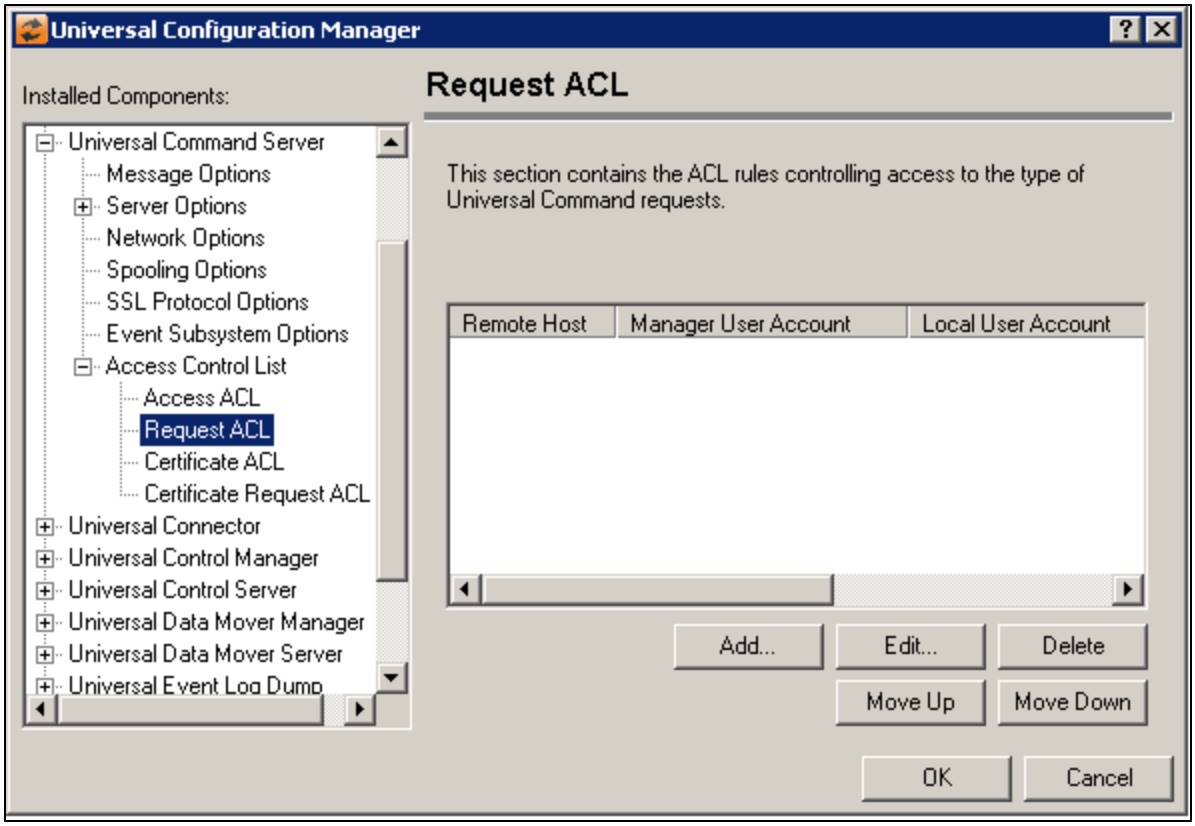


Field	Description

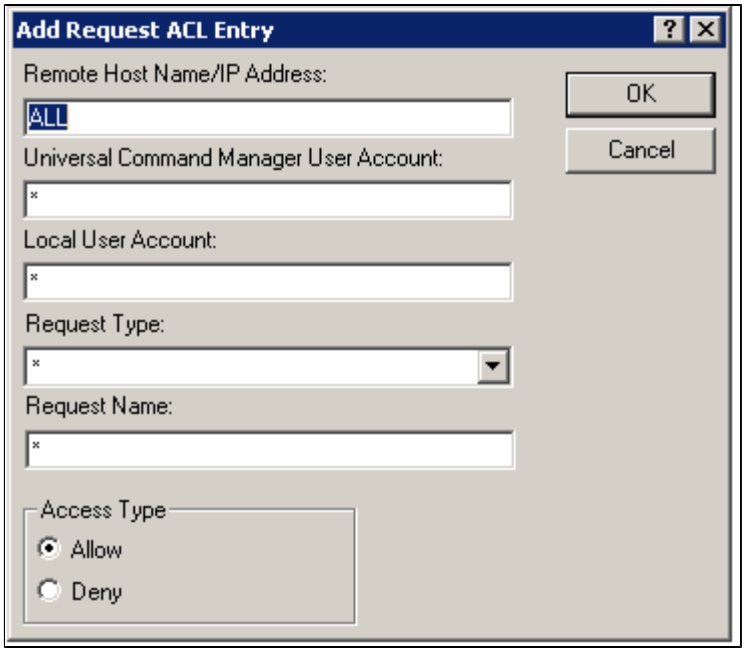
Access Control List	Displays the entries contained in the Universal Command Server Access Control List (ACL). This list is used to grant or deny access to Universal Command Server based on the remote system's IP address, the user account running the Universal Command Manager, and the user account with which the Server is executed. The order in which ACL entries are evaluated and applied are based on their position in this list. Use the Move Up and Move Down buttons to change an entry's priority.
Add Button	Displays a dialog box that allows an entry to be added to the Access Control List. The fields in this dialog are populated with default values.  **Remote Host Name/IP Address** The IP address or name of the remote host for which access will be allowed or denied. To apply an access control entry to all IP addresses, use the keyword ALL (in all caps). **Universal Command Manager User Account** (No Help provided.) **Local User Account** The local user account that will be used to run the requested process. **Access Type** Will cause the Universal Command Server to accept/reject incoming connections from the specified host, provided the Universal Command Manager is executed by the specified user account and the process is owned by the specified local user account.
Edit Button	Displays a dialog that allows the selected Access Control List entry to be modified.  **Remote Host Name/IP Address** The IP address or name of the remote host for which access will be allowed or denied. To apply an access control entry to all IP addresses, use the keyword ALL (in all caps). **Universal Command Manager User Account** (No Help provided.)

	Local User Account The local user account that will be used to run the requested process. **Access Type** Will cause the Universal Command Server to accept/reject incoming connections from the specified host, provided the Universal Command Manager is executed by the specified user account and the process is owned by the specified local user account.
Delete Button	Removes the selected item from the Access Control List.
Move Up Button	Moves the selected item up one position in the list.
Move Down Button	Moves the selected item down one position in the list.

Universal Command Server: Access Control List: Request ACL



Field	Description
Request ACL List	Displays the entries contained in the Universal Command Request Access Control List (ACL). The order in which ACL entries are evaluated and applied are based on their position in this list. Use the Move Up and Move Down buttons to change an entry's priority.
Add Button	Displays a dialog box that allows an entry to be added to the Access Control List. The fields in this dialog are populated with default values.



Remote Host Name/IP Address
The IP address or name of the remote host for which access will be allowed or denied. To apply an access control entry to all IP addresses, use the keyword ALL (in all caps).

Universal Command Manager User Account
The ID of the user account executing the Universal Command Manager.

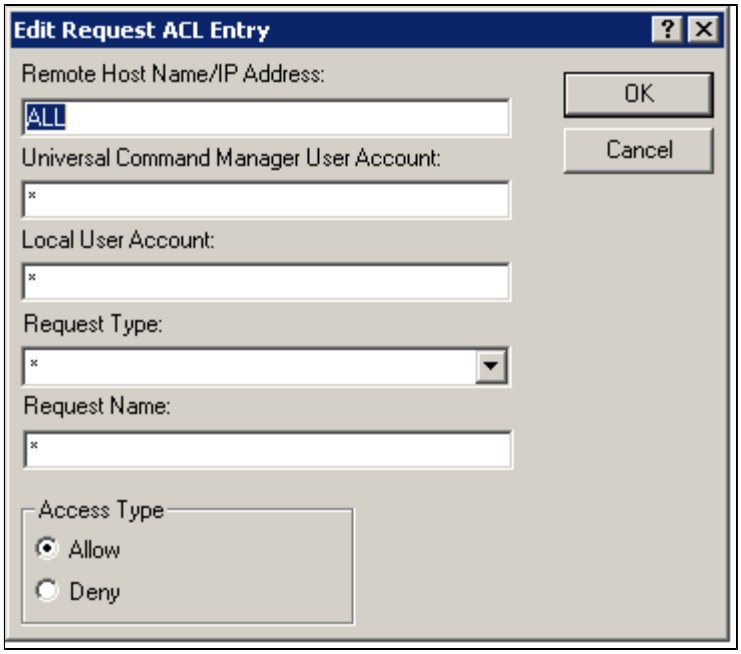
Local User Account
Specifies the name of the local user account to which this ACL entry will be applied. An entry of * indicates that the entry applies to all local user accounts.

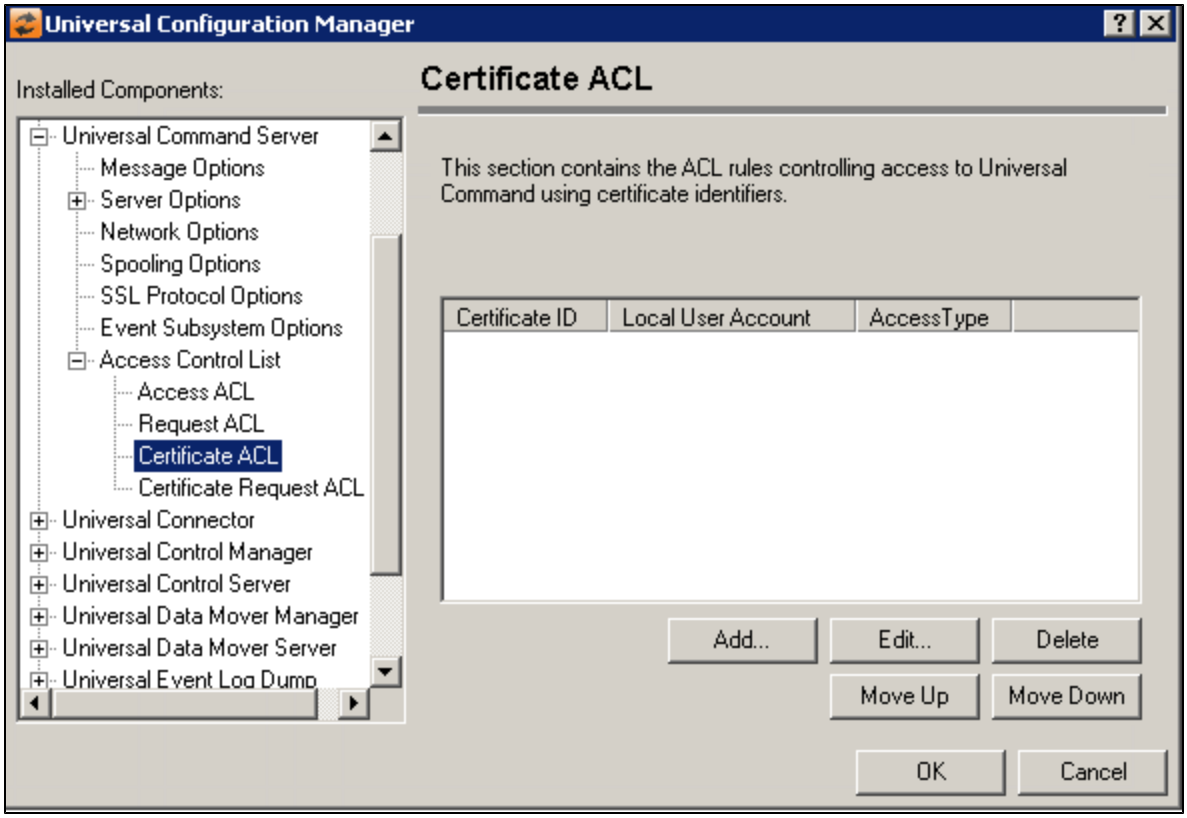
Request Type
Specifies the type of request issued by the UCmd Manager. Valid request types for UCmd Servers running on Windows are included in the list. * may also be specified to indicate the rule applies to all valid UCmd Manager request types.

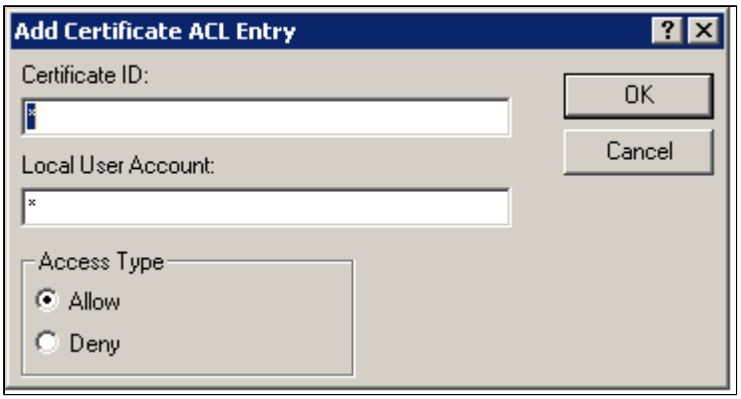
Request Name
Enter the name of a command or a command reference here to set access for the specified local user account based on the UCmd Manager request (as specified by the -cmd command line option). Use the default value of * to grant or deny access to all commands requested by the user.

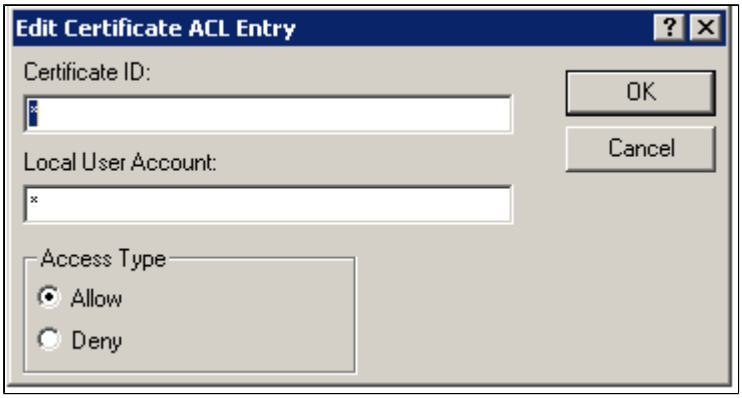
Access Type
Will cause the Universal Command Server to accept/reject incoming connections from the specified host, provided the Universal Command Manager is executed by the specified user account and the process is owned by the specified local user account.

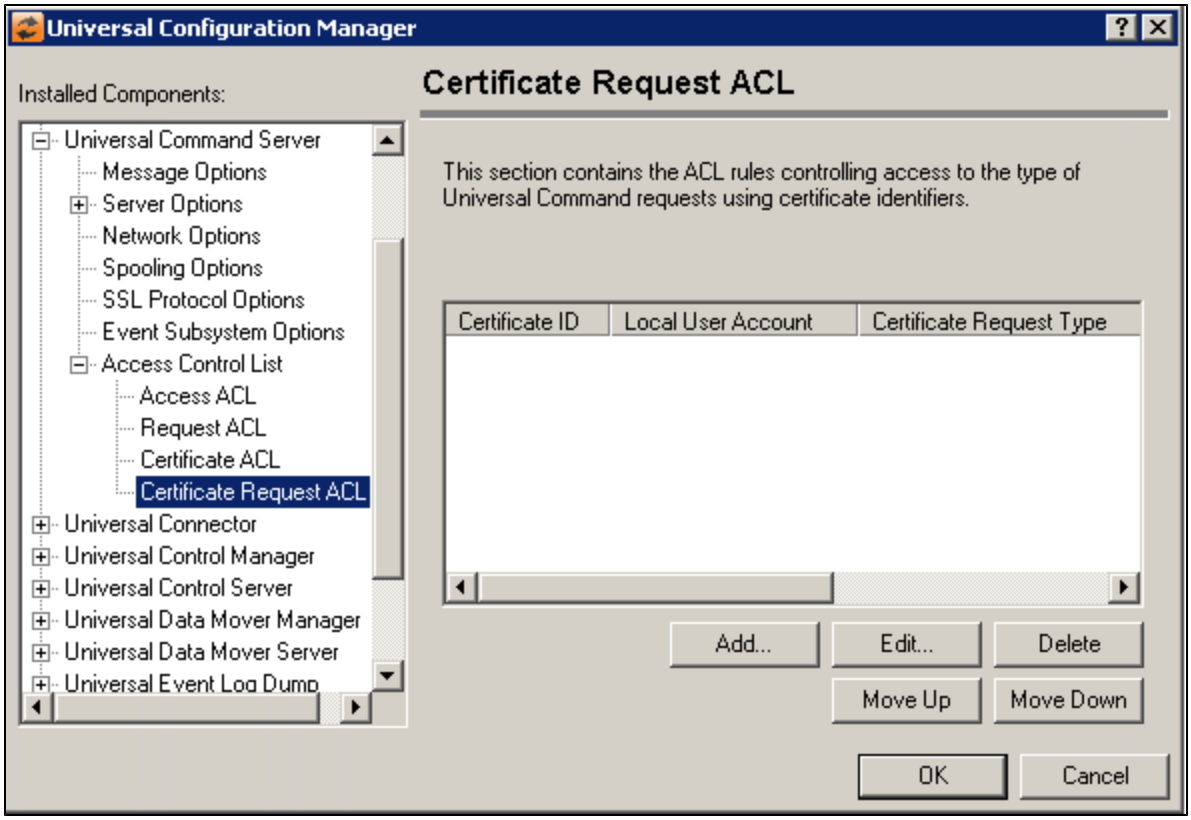
Edit Button	Displays a dialog that allows the selected Access Control List entry to be modified.

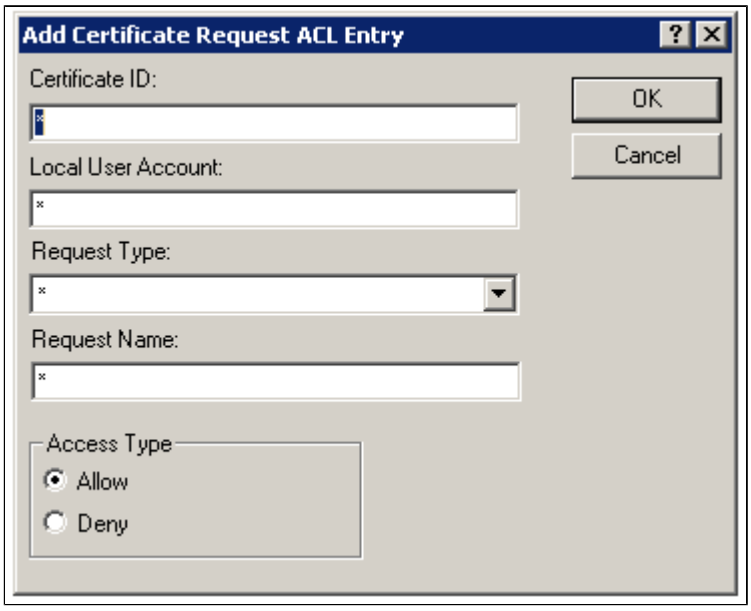
	 **Remote Host Name/IP Address** The IP address or name of the remote host for which access will be allowed or denied. To apply an access control entry to all IP addresses, use the keyword ALL (in all caps). **Universal Command Manager User Account** The ID of the user account executing the Universal Command Manager. **Local User Account** Specifies the name of the local user account to which this ACL entry will be applied. An entry of * indicates that the entry applies to all local user accounts. **Request Type** Specifies the type of request issued by the UCmd Manager. Valid request types for UCmd Servers running on Windows are included in the list. * may also be specified to indicate the rule applies to all valid UCmd Manager request types. **Request Name** Enter the name of a command or a command reference here to set access for the specified local user account based on the UCmd Manager request (as specified by the -cmd command line option). Use the default value of * to grant or deny access to all commands requested by the user. **Access Type** Will cause the Universal Command Server to accept/reject incoming connections from the specified host, provided the Universal Command Manager is executed by the specified user account and the process is owned by the specified local user account.
Delete Button	Removes the selected item from the Access Control List.
Move Up Button	Moves the selected item up one position in the list.
Move Down Button	Moves the selected item down one position in the list.

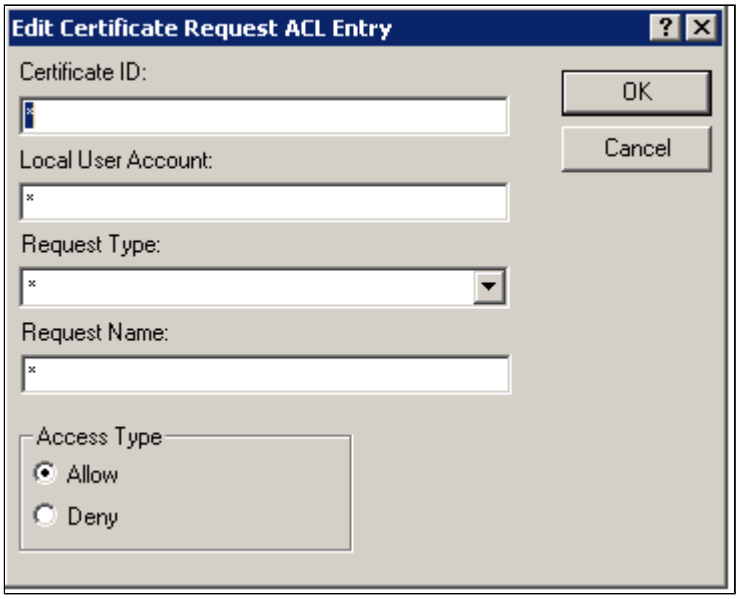


Field	Description
Certificate ACL List	Displays the entries contained in the Universal Command Certificate Access Control List (ACL). The order in which ACL entries are evaluated and applied are based on their position in this list. Use the Move Up and Move Down buttons to change an entry's priority.
Add Button	Displays a dialog box that allows an entry to be added to the Access Control List. The fields in this dialog are populated with default values.  **Certificate ID** When a UCmd Manager specifies a certificate ID from its command line, its corresponding UCmd Server compares it to the certificate ID entered here. If a match is found and the certificate ID has a corresponding entry in the Certificate Map list, the authority to execute the request is granted or denied to the specified local user account. **Local User Account** The local user account that will be used to run the requested process. **Access Type** Will cause the Universal Command Server to accept/reject incoming connections from the specified host, provided the Universal Command Manager is executed by the specified user account and the process is owned by the specified local user account.

Edit Button	Displays a dialog that allows the selected Access Control List entry to be modified.  **Certificate ID** When a UCmd Manager specifies a certificate ID from its command line, its corresponding UCmd Server compares it to the certificate ID entered here. If a match is found and the certificate ID has a corresponding entry in the Certificate Map list, the authority to execute the request is granted or denied to the specified local user account. **Local User Account** The local user account that will be used to run the requested process. **Access Type** Will cause the Universal Command Server to accept/reject incoming connections from the specified host, provided the Universal Command Manager is executed by the specified user account and the process is owned by the specified local user account.
Delete Button	Removes the selected item from the Access Control List.
Move Up Button	Moves the selected item up one position in the list.
Move Down Button	Moves the selected item down one position in the list.



Field	Description
Certificate Request ACL List	Displays the entries contained in the Universal Command Certificate Request Access Control List (ACL). The order in which ACL entries are evaluated and applied are based on their position in this list. Use the Move Up and Move Down buttons to change an entry's priority.
Add Button	Displays a dialog box that allows an entry to be added to the Access Control List. The fields in this dialog are populated with default values.  **Certificate ID** When a UCmd Manager specifies a certificate ID from its command line, its corresponding UCmd Server compares it to the certificate ID entered here. If a match is found and the certificate ID has a corresponding entry in the Certificate Map list, the authority to execute the request is granted or denied to the specified local user account.

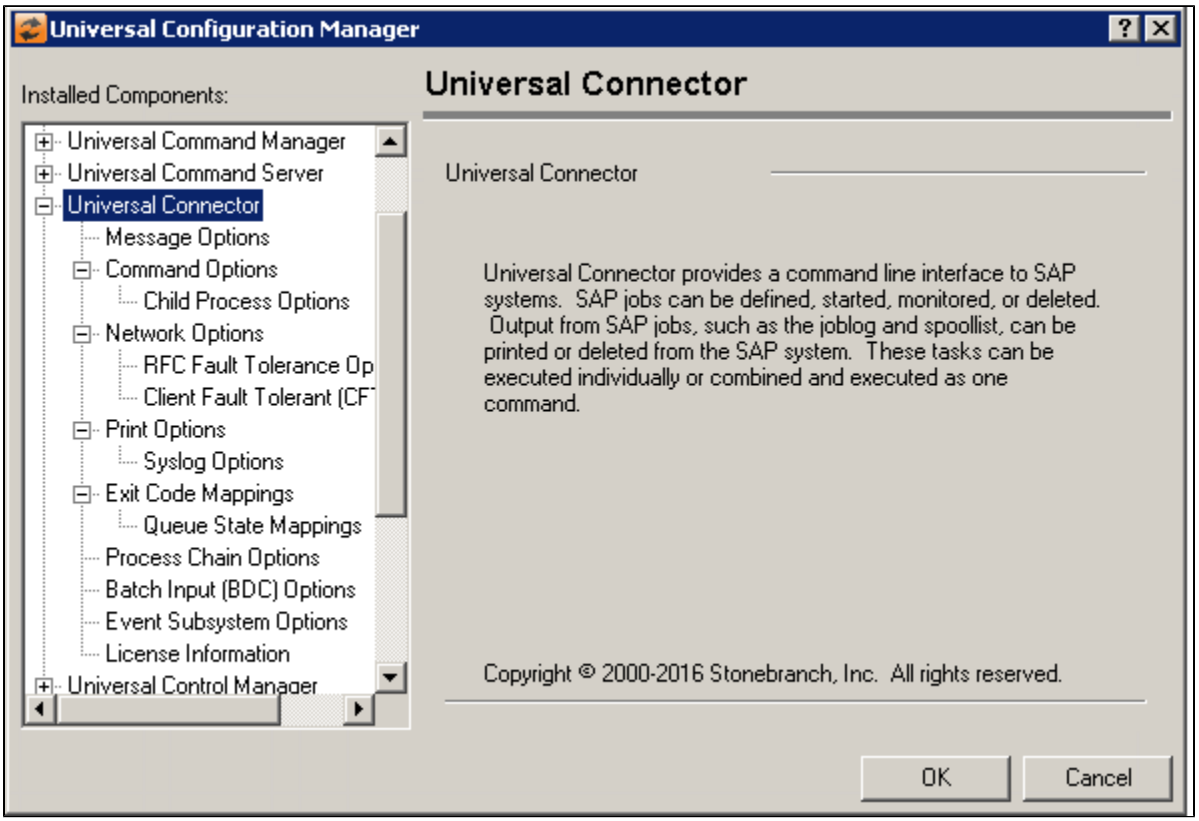
	Local User Account The local user account that will be used to run the requested process. **Request Type** Specifies the type of request issued by the UCmd Manager. Valid request types for UCmd Servers running on Windows are included in the list. * may also be specified to indicate the rule applies to all valid UCmd Manager request types. **Request Name:** Enter the name of a command or a command reference here to set access for the specified local user account based on the UCmd Manager request (as specified by the -cmd command line option). Use the default value of * to grant or deny access to all commands requested by the user. **Access Type** Will cause the Universal Command Server to accept/reject incoming connections from the specified host, provided the Universal Command Manager is executed by the specified user account and the process is owned by the specified local user account.
Edit Button	Displays a dialog that allows the selected Access Control List entry to be modified.  **Certificate ID** When a UCmd Manager specifies a certificate ID from its command line, its corresponding UCmd Server compares it to the certificate ID entered here. If a match is found and the certificate ID has a corresponding entry in the Certificate Map list, the authority to execute the request is granted or denied to the specified local user account. **Local User Account** The local user account that will be used to run the requested process. **Request Type** Specifies the type of request issued by the UCmd Manager. Valid request types for UCmd Servers running on Windows are included in the list. * may also be specified to indicate the rule applies to all valid UCmd Manager request types. **Request Name:** Enter the name of a command or a command reference here to set access for the specified local user account based on the UCmd Manager request (as specified by the -cmd command line option). Use the default value of * to grant or deny access to all commands requested by the user. **Access Type** Will cause the Universal Command Server to accept/reject incoming connections from the specified host, provided the Universal Command Manager is executed by the specified user account and the process is owned by the specified local user account.
Delete Button	Removes the selected item from the Access Control List.
Move Up Button	Moves the selected item up one position in the list.
	Moves the selected item down one position in the list.

Move Down Button	

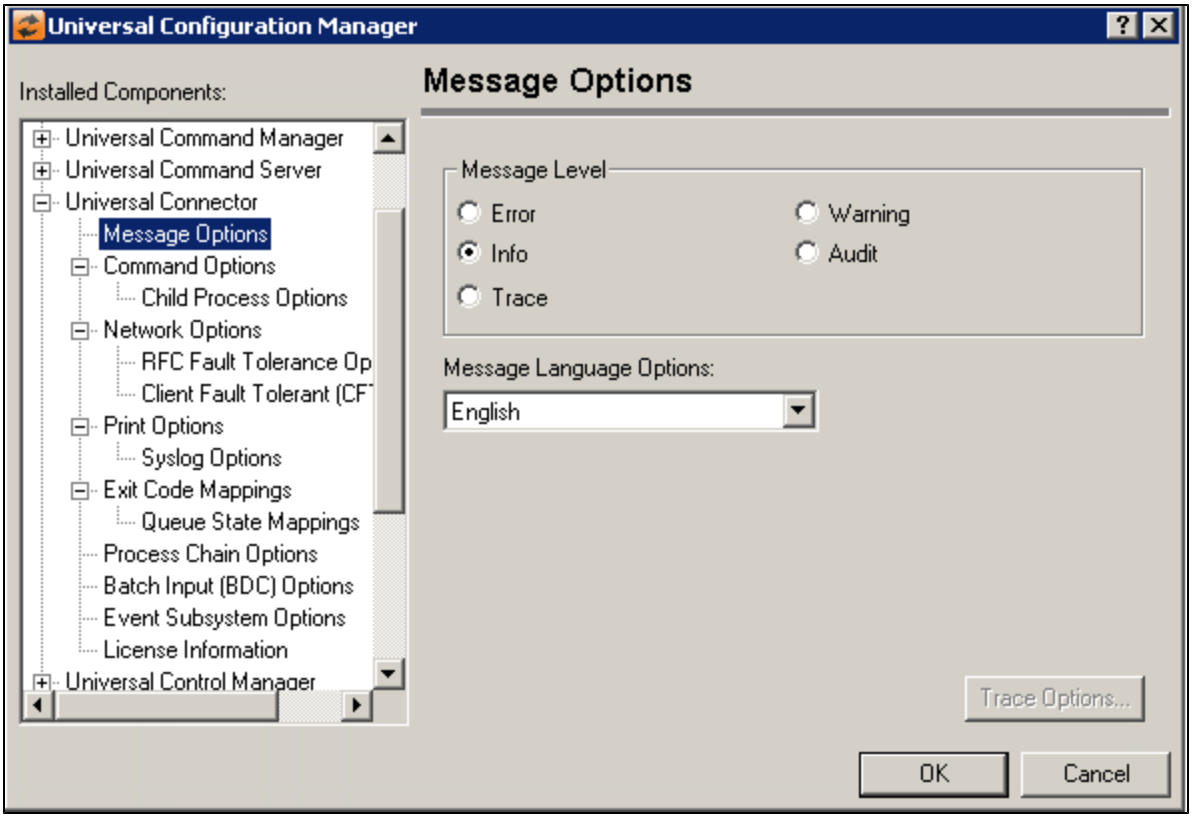
Universal Configuration Manager - Universal Connector for SAP

- Universal Connector
- Universal Connector: Message Options
- Universal Connector: Command Options
  - Universal Connector: Command Options - Child Process Options
- Universal Connector: Network Options
  - Universal Connector: Network Options - RFC Fault Tolerance Options
  - Universal Connector: Network Options - Client Fault Tolerance (CFT) Options
- Universal Connector: Print Options
  - Universal Connector: Print Options - Syslog Options
- Universal Connector: Exit Code Mappings
  - Universal Connector: Exit Code Mappings - Queue State Mappings
- Universal Connector: Process Chain Options
- Universal Connector: Batch Input (BDC) Options
- Universal Connector: Event Subsystem Options
- Universal Connector: License Information

Universal Connector

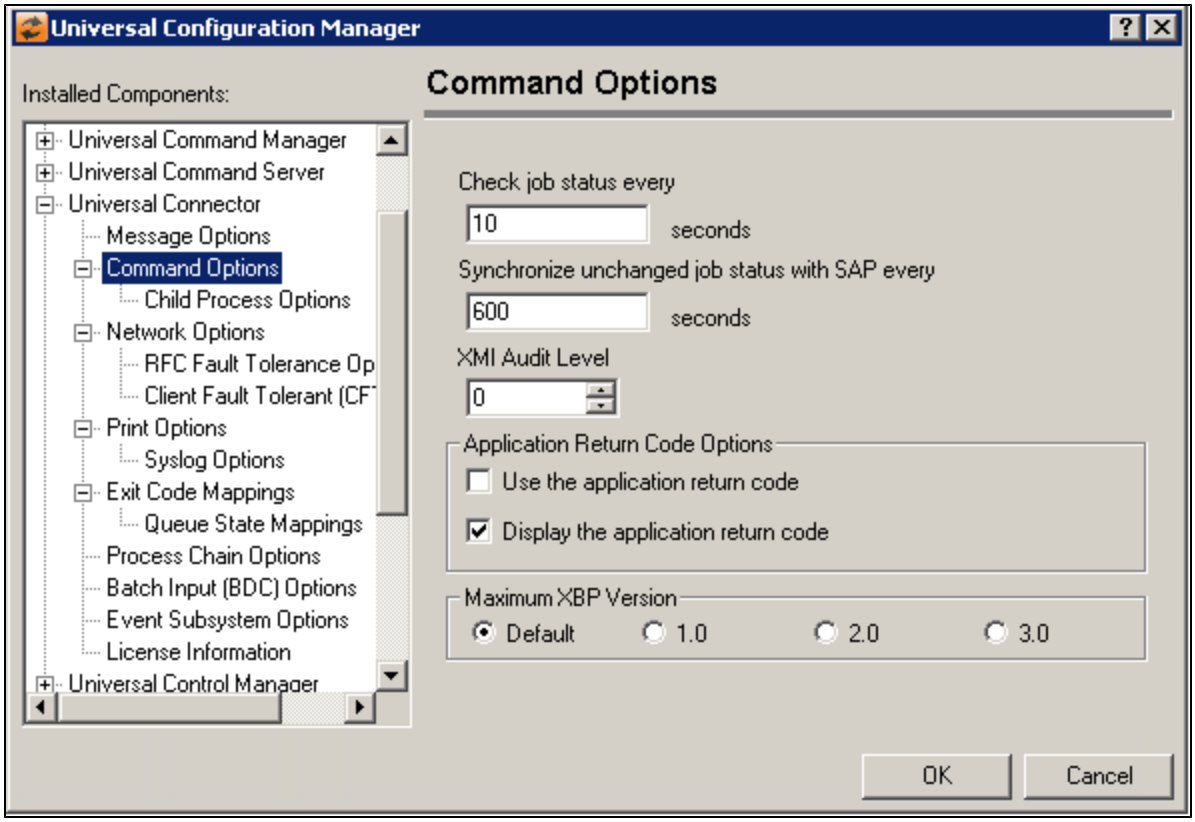


Universal Connector: Message Options

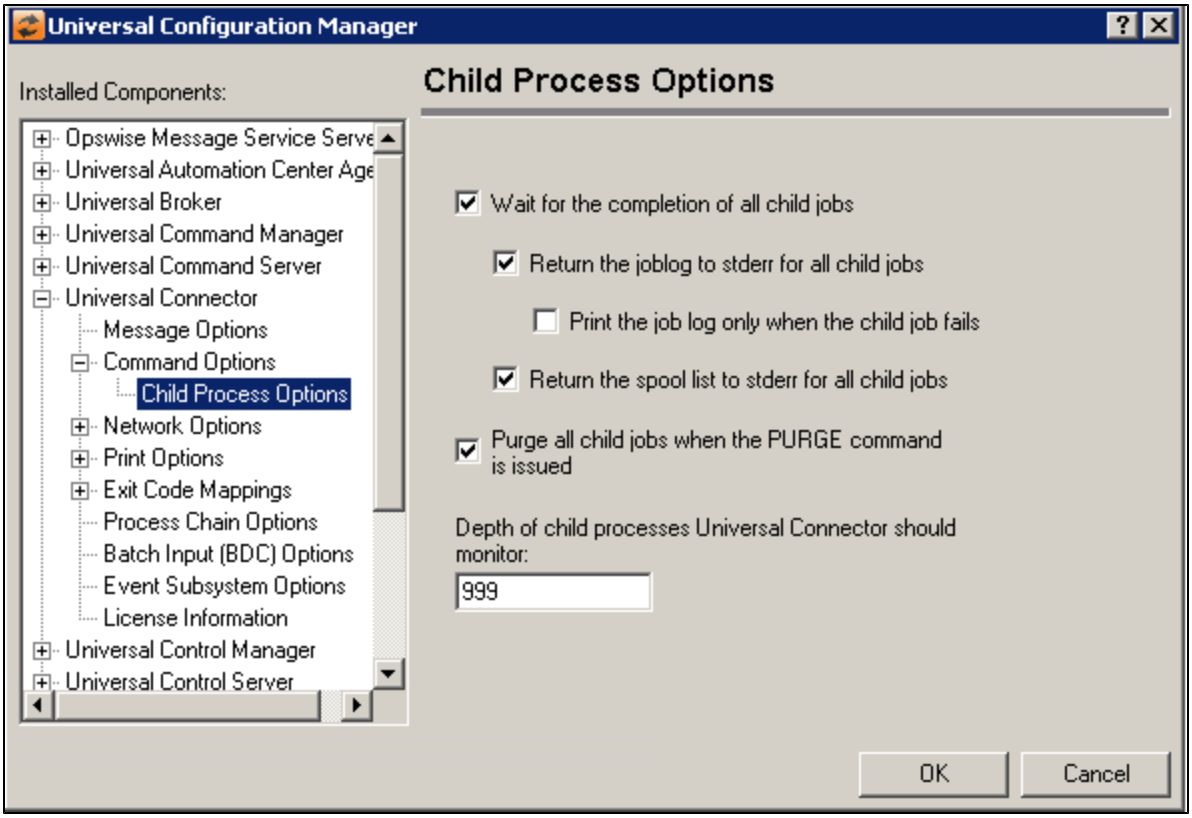


Field	Description
Message Level	**Error** Turns on message logging for errors only. **Warning** Turns on message logging for warnings and errors. **Info** Turns on message logging for all informational, warning and error messages. **Audit** Turns on message logging for all audit, informational, warning and error messages. **Trace** Allows all messages to be logged, including those used for debugging purposes.
Message Language Options	Specifies the language used to display error messages.

Universal Connector: Command Options

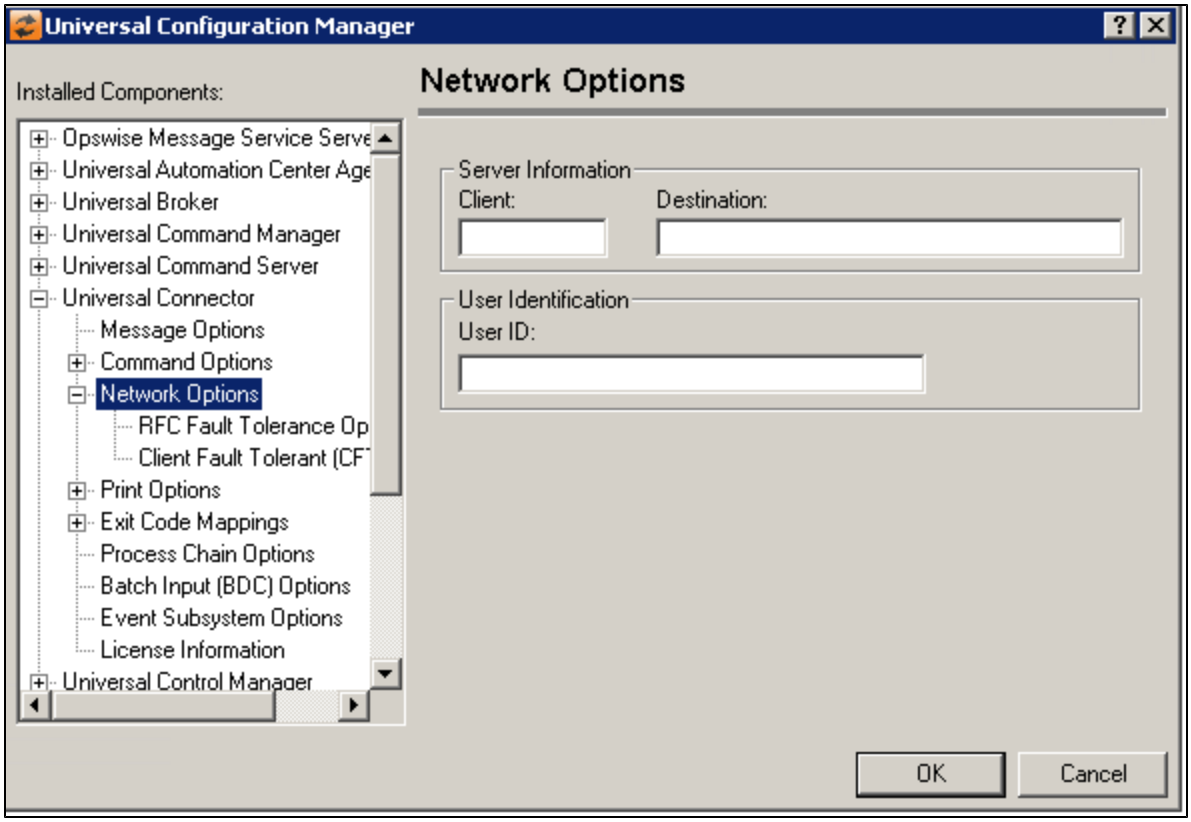


Field	Description
Check job status every	This value specifies how often the status of the job is checked when the RUN command is issued, or when the WAIT option is used with the START command. A value of 0 (zero) causes the job status to be checked repeatedly, with no wait between checks. The default value for this option is 10 seconds. This option sets the value for the poll_time configuration option.
Synchronize unchanged job status with SAP every	Specifies the number of seconds that can elapse with no job status changes, before Universal Connector will synchronize the job's actual status with the status stored in SAP. This default value for this option is 600 seconds. This option sets the value for the job_stat_check_interval configuration option.
XMI Audit level	Determines the degree of information written to the XMI log. The amount of information increases with higher audit level values. The default value for this option is 0. This option sets the value for the xmi_audit_level configuration option.
Application Return Code Options	**Use the application return code** When this option is checked, Universal Connector sets its exit code based upon a job's application return code. This option is not checked by default. This option sets the value for the use_app_rc configuration option. **Display the application return code** When this option is checked, Universal Connector returns any application return codes set for a job to stderr. This option is checked by default. This option sets the value for the print_app_rc configuration option.
Maximum XBP Version	Specifies the maximum version of the SAP XBP interface that Universal Connector will use during execution. When Default is selected, Universal Connector uses the highest level supported by the SAP system. This option sets the value for the max_xbp configuration option.



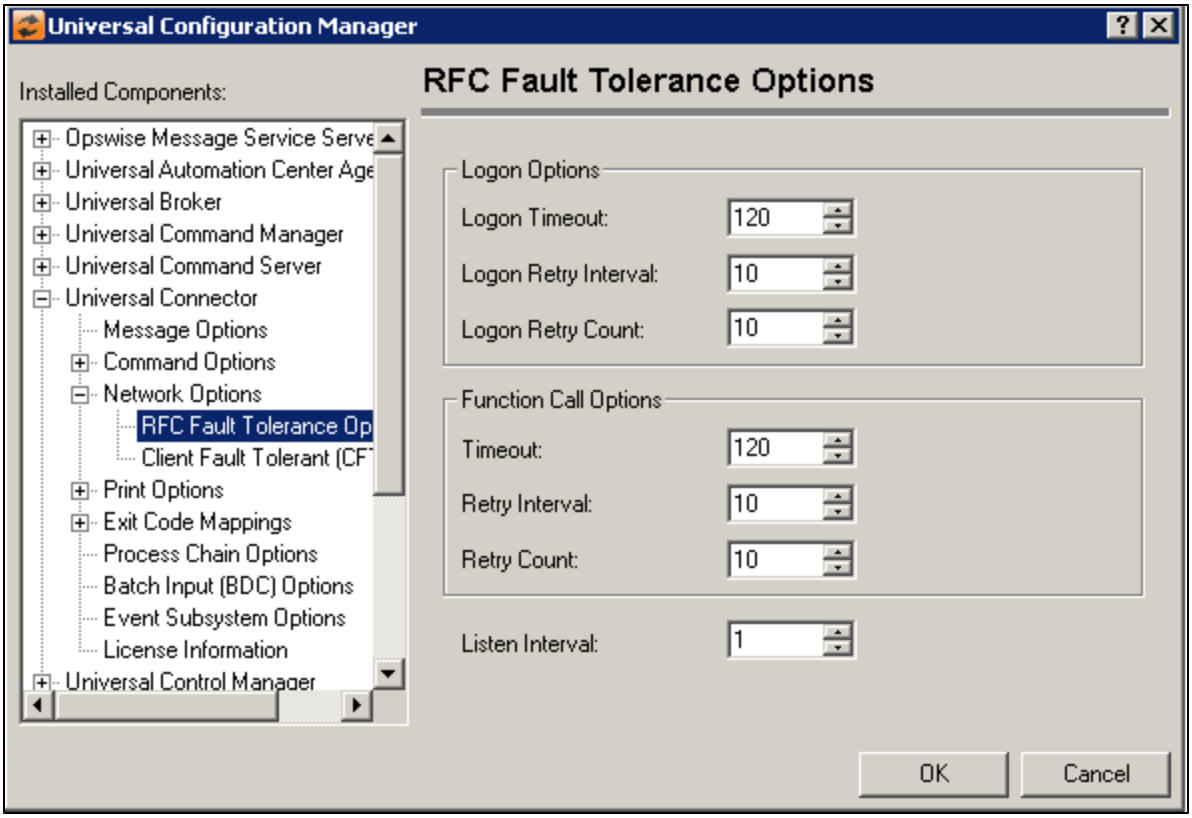
Field	Description
Wait for completion of all child jobs	When this option is checked, Universal Connector will monitor all child jobs and wait for their completion. Universal Connector will exit with the most significant completion status received from all monitored jobs. This option is checked by default. This option sets the value for the wait_for_child_jobs configuration option.
Return the job log to stderr for all child jobs	When this option is checked, Universal Connector will print the job logs for child jobs to stderr. This option is checked by default, but is recognized only when Universal Connector is configured to wait for child jobs AND to return job logs for parent processes. This option sets the value for the print_joblog_for_child_jobs configuration option to yes (the default) or no.
Print the job log only when the child job fails	When this option is checked, Universal Connector will only return the job logs for child jobs that did not complete successfully. This option is not checked by default. When this option is checked, the value of the print_joblog_for_child_jobs option is set the error.
Return the spool list to stderr for all child jobs	When this option is checked, Universal Connector will print the spool lists for child jobs to stderr. This option is checked by default, but is recognized only when Universal Connector is configured to wait for child jobs AND to return spool lists for parent processes. This option sets the value for the print_spoollist_for_child_jobs configuration option.
Purge all child jobs when the PURGE command is issued	When this option is checked, Universal Connector will purge all child jobs from the SAP system whenever the PURGE command is issued. This option is checked by default. This option sets the value for the purge_child_jobs configuration option.
Depth of child processes Universal Connector should monitor	Specifies the maximum depth of child processes that Universal Connector will monitor. When set to 1, Universal Connector will only check for an monitor jobs created by the initial parent job. To monitor multiple "generations" of child processes, specify a value greater than 1, up to a maximum of 999. The default value for this option is 999. This option sets the value for the max_child_depth configuration option.

Universal Connector: Network Options

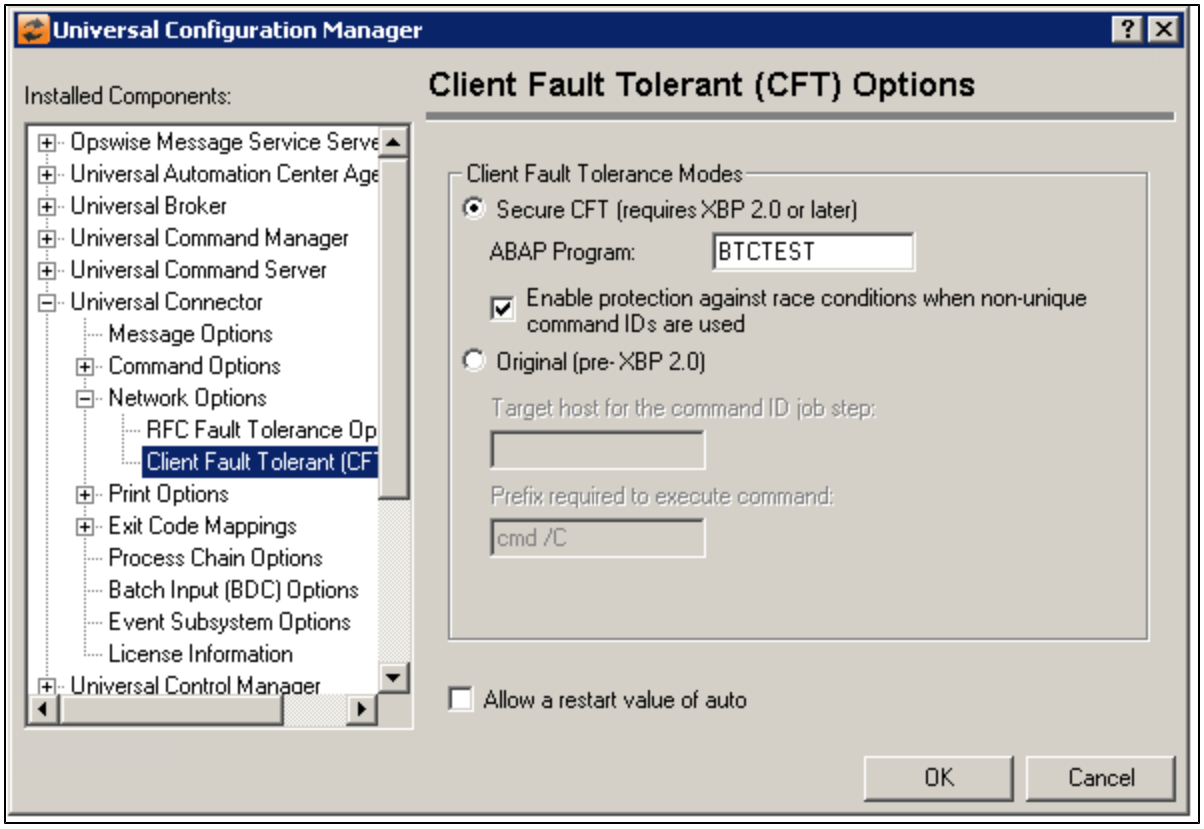


Field	Description
Server Information	**Client** The SAP client number. **Destination** The name of the SAP host, as specified in the saprfc.ini file.
User Identification	The remote SAP user ID that will be used to execute commands.

Universal Connector: Network Options - RFC Fault Tolerance Options

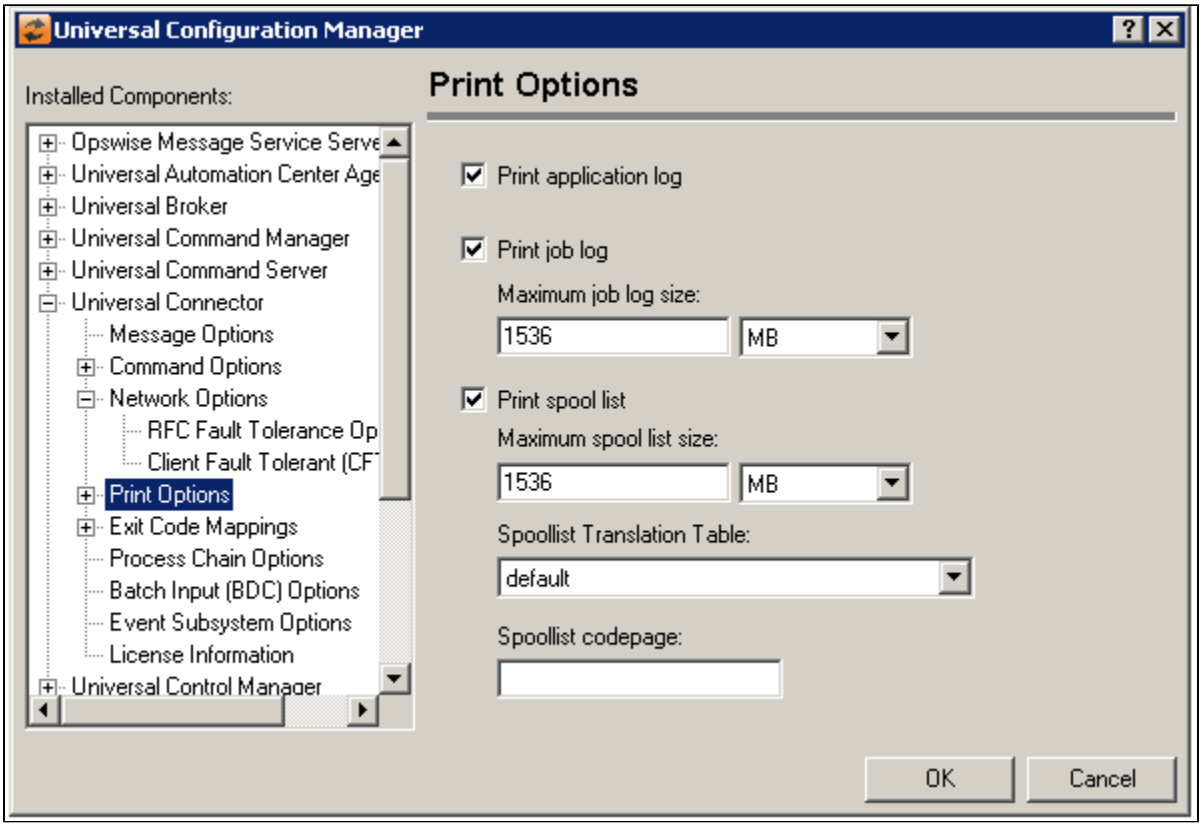


Field	Description
Logon Options	**Logon Timeout** Sets a time constraint on the RFC logon process. The number of seconds that can elapse before USAP considers the RFC logon process to have timed out. **Logon Retry Interval** Specifies the number of seconds that will elapse between a failed RFC logon attempt and the retry of that logon attempt. **Logon Retry Count** Specifies the number of unsuccessful RFC logon retry attempts that can occur before USAP terminates the logon process and ends unsuccessfully.
Function Call Options	**Timeout** Sets time constraint on all RFC functions, with the exception of blocking connection calls. The number of seconds that can elapse before USAP considers an RFC call to have timed out. **Retry Interval** Specifies the number of seconds that will elapse between a failed RFC call and the retry of that RFC call. **Retry Count** Specifies the number of unsuccessful RFC call retry attempts that can occur before USAP terminates the RFC call retry process and ends unsuccessfully.
Listen Interval	Specifies the number of seconds that will elapse between RFC listen calls. These are polling calls that are performed repetitively to determine if an RFC event is available.



Field	Description
Client Fault Tolerance Modes	**Secure CFT (Requires XBP 2.0 or later)** When this option is checked, enables secure client fault tolerance (CFT). This option requires XBP 2.0 or later. This option is checked by default. This option sets the value for the cft_secure_cft configuration option. **ABAP Program** Specifies the ABAP program to use for the command ID job step when secure client fault tolerance (CFT) is used. The default value for this option is BTCTEST. This option sets the value for the cft_abap configuration option. **Enable protection against race conditions when non-unique command ID's are used** When this option is checked, this option will prevent race conditions that could occur when multiple Universal Connector instances executing at the same time use secure client fault tolerance (CFT) and identical command ID parameters. This option is checked by default. This option sets the value for the cft_race_guard configuration option. **Original (pre-XBP 2.0)** When this option is checked, causes Universal Connector to use the original, pre-XBP 2.0 client fault tolerance (CFT) logic. This option is not checked by default. This option sets the value for the cft_secure_cft configuration option. **Target Host for the command ID job step** Specifies the system Universal Connector uses for its Command ID job step when the Command ID option is used. This option has no default value. This option sets the value for the cft_target_host configuration option. **Prefix required to execute command** Specifies the prefix required to issue a command on the target host. Universal Connector only recognizes this option when a command ID is specified. This default value for this option is cmd /C. This option sets the value for the cft_cmd_prefix configuration option.
Allow a restart value of auto	When this option is checked, Universal Connector will accept a value of "auto" for the -restart option. This option is checked by default. This option sets the value for the auto_restart_ok configuration option.

Universal Connector: Print Options



Field	Description
Print Application Log	When this option is checked, Universal Connector will return a job's application log to stderr. This option is checked by default. This option sets the value for the print_applog configuration option.
Print job log Options	**Print job log** When this option is checked, Universal Connector will send a job's job log to standard out. This option is checked by default. This option sets the value for the print_joblog configuration option. **Maximum job log size** Specifies the maximum size for job logs. Universal Connector will not transfer any job logs that exceed this size. Maximum allowable values are: • Bytes 2,147,483,647 • KB 2,097,151 • MB 2,047 The default value for this option is 1536 MB. This option sets the value for the max_log_size configuration option.
Print spool list options	**Print Spool List** When this option is checked, the spool list will be sent to standard output. This option is checked by default. This option sets the value for the print_spoollist configuration option. **Maximum spool list size** Specifies the maximum size for spool lists. Universal Connector will not transfer any spool lists that exceed this size. Maximum allowable values are: • Bytes 2,147,483,647

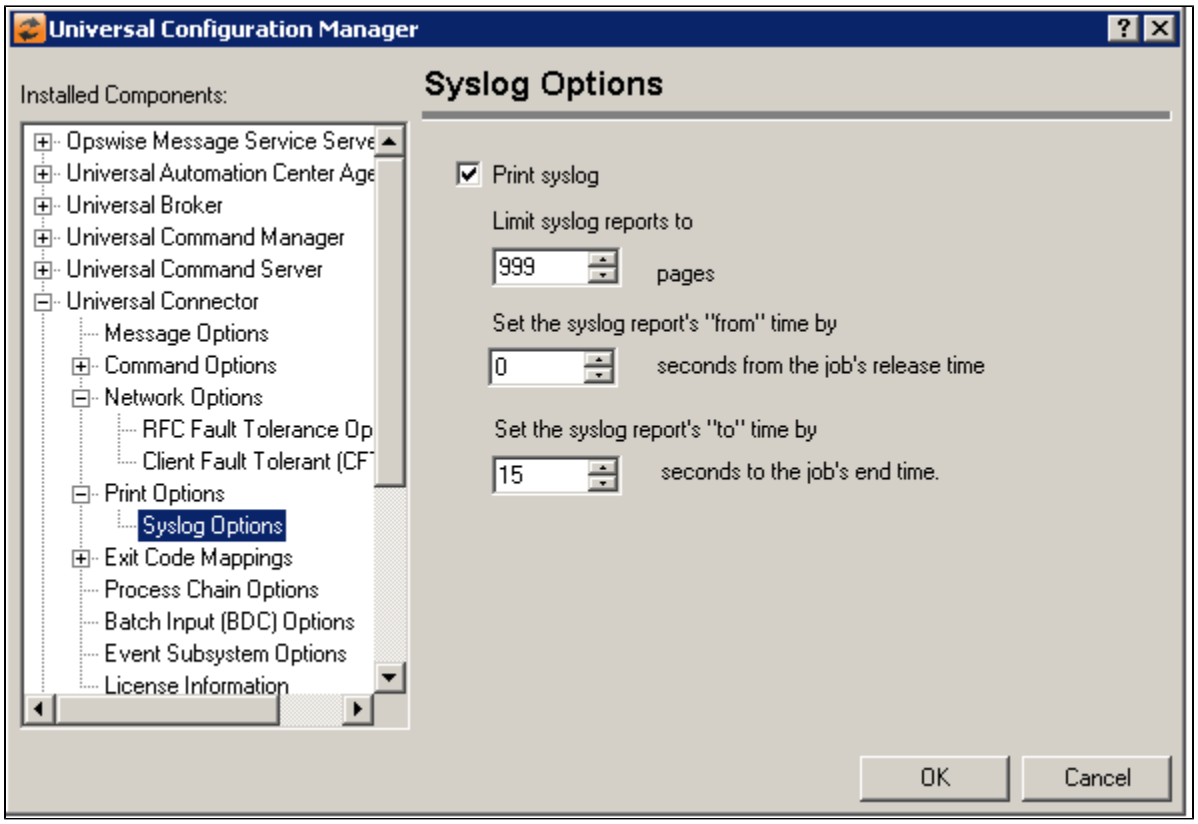
- KB
  2,097,151
- MB
  2,047

This default value for this option is 1536 MB.
This option sets the value for the max_spool_size configuration option.

Spool Translation Table
The translation table specifies the Spool List Translation Table (.stt) file to use for spool list formatting.
The default Spool List Translation Table file, default, contains translations for the standard SAP formatting controls into appropriate display and control characters. Translation Table files are located in the nls subdirectory of the installation directory. The value must specify a file located in the nls directory without its stt extension.
This option sets the value for the translation table configuration option.

Spool list codepage
Specifies the codepage Universal Connector uses to transfer spool lists from the SAP system.
There is no default value for this option.
This option sets the value for the spool_codepage configuration option.

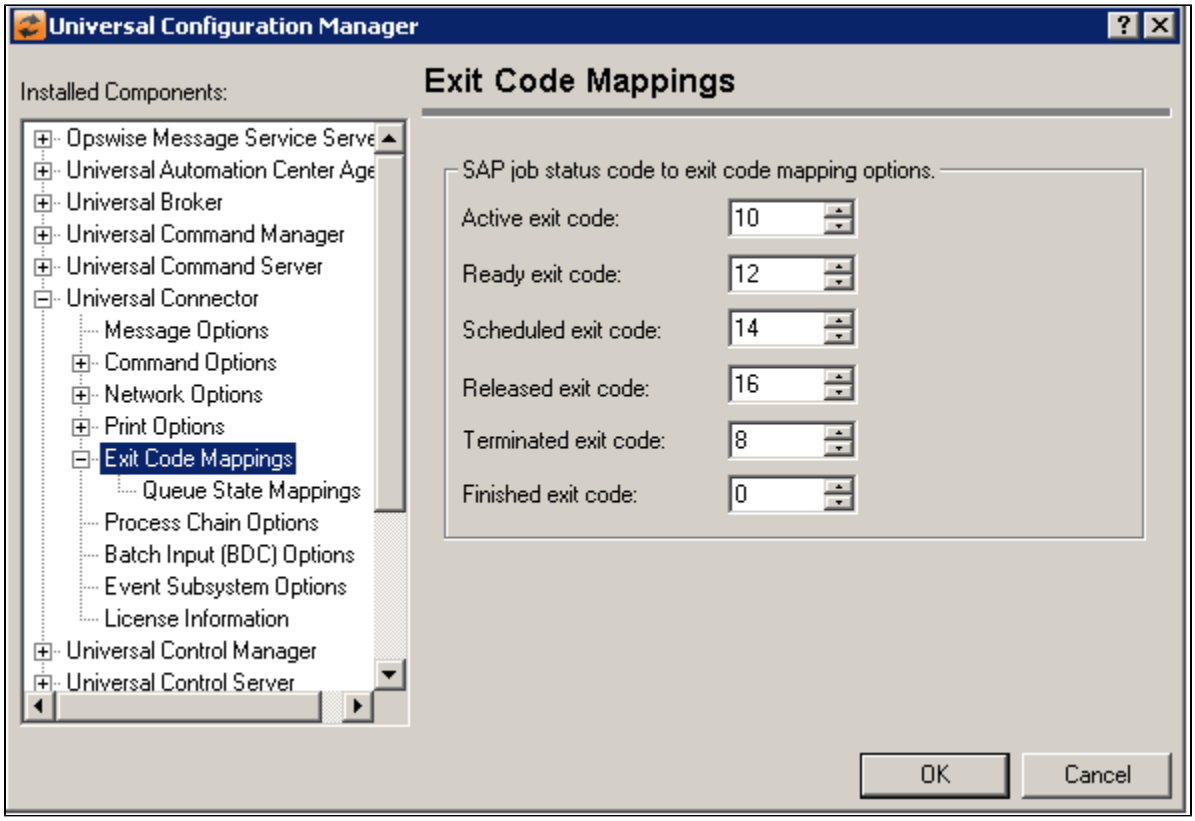
Universal Connector: Print Options - Syslog Options



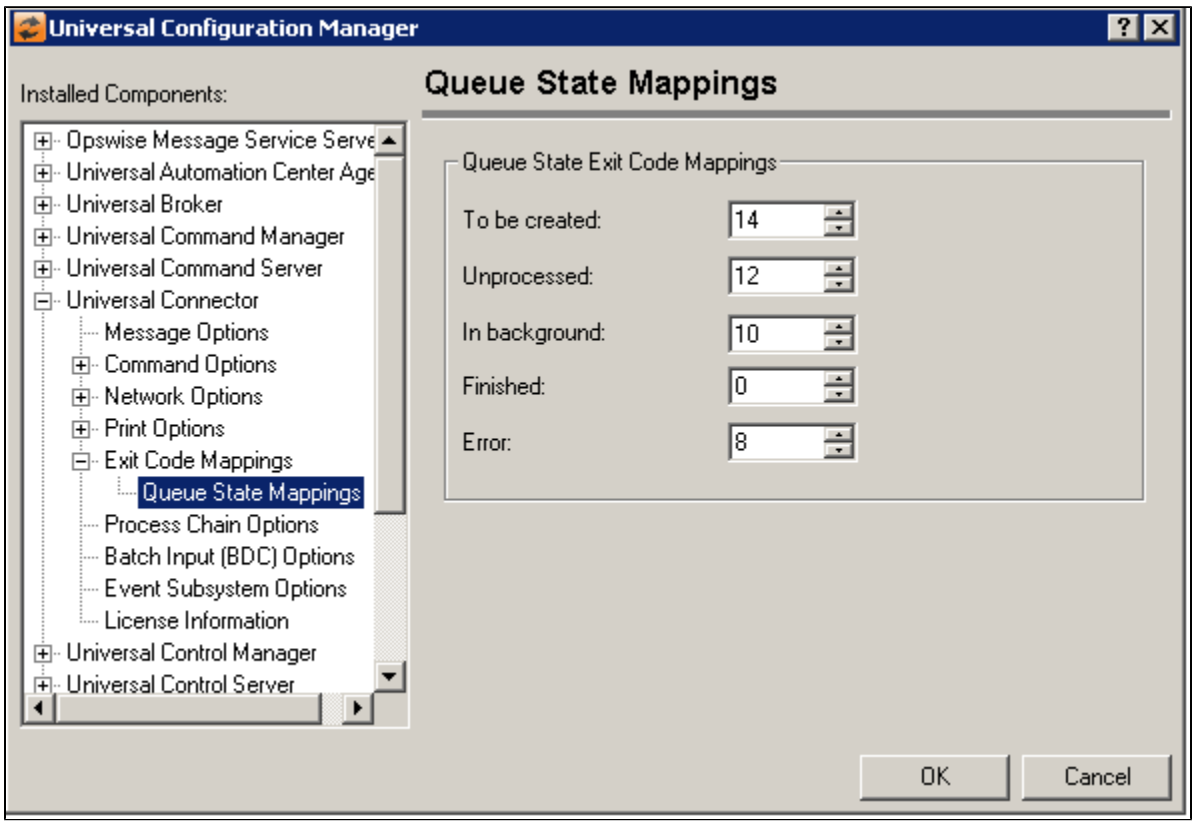
Field	Description
Print syslog	When this option is checked, Universal Connector will generate a syslog report for a job and display it on stderr. This option is checked by default. This option sets the value for the print_syslog configuration option.
Limit Syslog reports to	Specifies the number of seconds Universal Connector will subtract from the job's release time when calculating the from time of a syslog report. The default value for this option is 0. This option sets the value for the syslog_pre_time configuration option.
Set the syslog report's "from" time by	Specifies the number of seconds Universal Connector will add to the job's end time when calculating the to time of a syslog report. The default value for this option is 15. This option sets the value for the syslog_post_time configuration option.

Set the syslog report's "to" time by	Specifies the number of seconds Universal Connector will add to the job's end time when calculating the to time of a syslog report. The default value for this option is 15. This option sets the value for the syslog_post_time configuration option.

Universal Connector: Exit Code Mappings

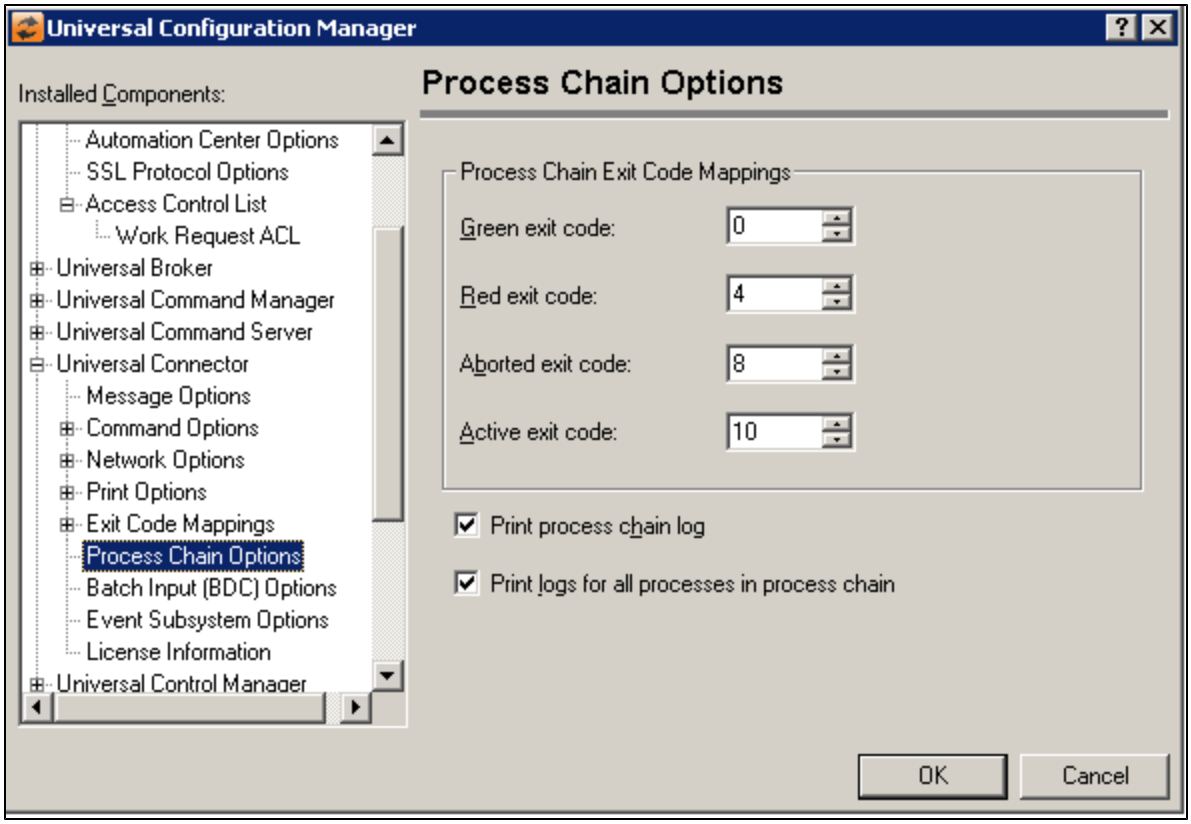


Field	Description
SAP Job Status code to exit code mapping options	**Active exit code** The usap program exits with the SAP job status code when usap is directed to check the status of a job or to wait until a job completes execution. The following options can be used to map SAP job status codes to specific usap program exit codes. The default exit codes are the SAP status codes. **Ready exit code** The usap program exits with the SAP job status code when usap is directed to check the status of a job or to wait until a job completes execution. The following options can be used to map SAP job status codes to specific usap program exit codes. The default exit codes are the SAP status codes. **Scheduled exit code** The usap program exits with the SAP job status code when usap is directed to check the status of a job or to wait until a job completes execution. The following options can be used to map SAP job status codes to specific usap program exit codes. The default exit codes are the SAP status codes. **Released exit code** The usap program exits with the SAP job status code when usap is directed to check the status of a job or to wait until a job completes execution. The following options can be used to map SAP job status codes to specific usap program exit codes. The default exit codes are the SAP status codes. **Terminated exit code** The usap program exits with the SAP job status code when usap is directed to check the status of a job or to wait until a job completes execution. The following options can be used to map SAP job status codes to specific usap program exit codes. The default exit codes are the SAP status codes. **Finished exit code** The usap program exits with the SAP job status code when usap is directed to check the status of a job or to wait until a job completes execution. The following options can be used to map SAP job status codes to specific usap program exit codes. The default exit codes are the SAP status codes.



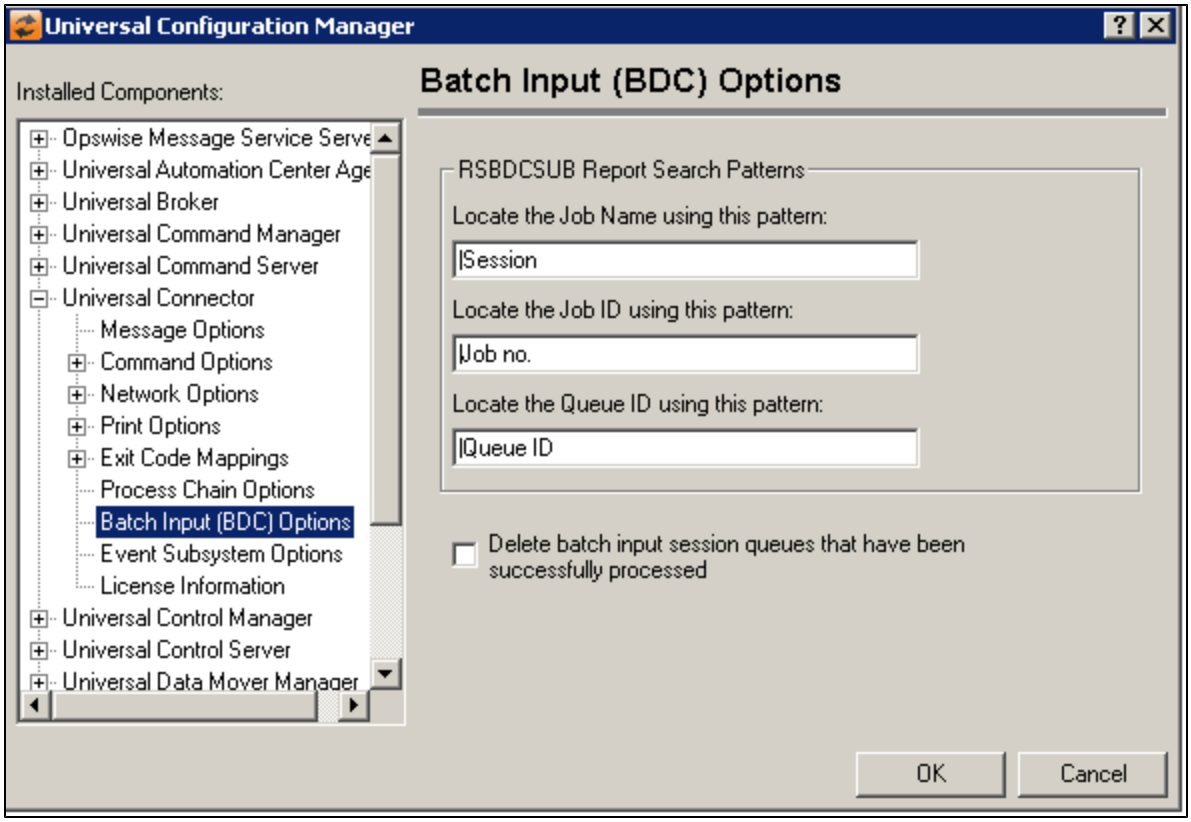
Field	Description
Queue State Exit Code Mappings	**To Be Created** Specifies the exit code to use when the result of the DISPLAY QSTATE command indicates that the queue's current state is "to be created". The default value for this option is 10. This option sets the value for the qtobecreated_exit_code configuration option. **Unprocessed** Specifies the exit code to use when the result of the DISPLAY QSTATE command indicates that the queue's current state is "unprocessed". The default value for this option is 12. This option sets the value for the qunprocessed_exit_code configuration option. **In background** Specifies the exit code to use when the result of the DISPLAY QSTATE command indicates that the queue's current state is "in background". The default value for this option is 14. This option sets the value for the qinbackground_exit_code configuration option. **Finished** Specifies the exit code to use when the result of the DISPLAY QSTATE command indicates that the queue's current state is "finished". The default value for this option is 16. This option sets the value for the qfinished_exit_code configuration option. **Error** Specifies the exit code to use when the result of the DISPLAY QSTATE command indicates that the queue's current state is "error". The default value for this option is 18. This option sets the value for the qerror_exit_code configuration option.*

Universal Connector: Process Chain Options



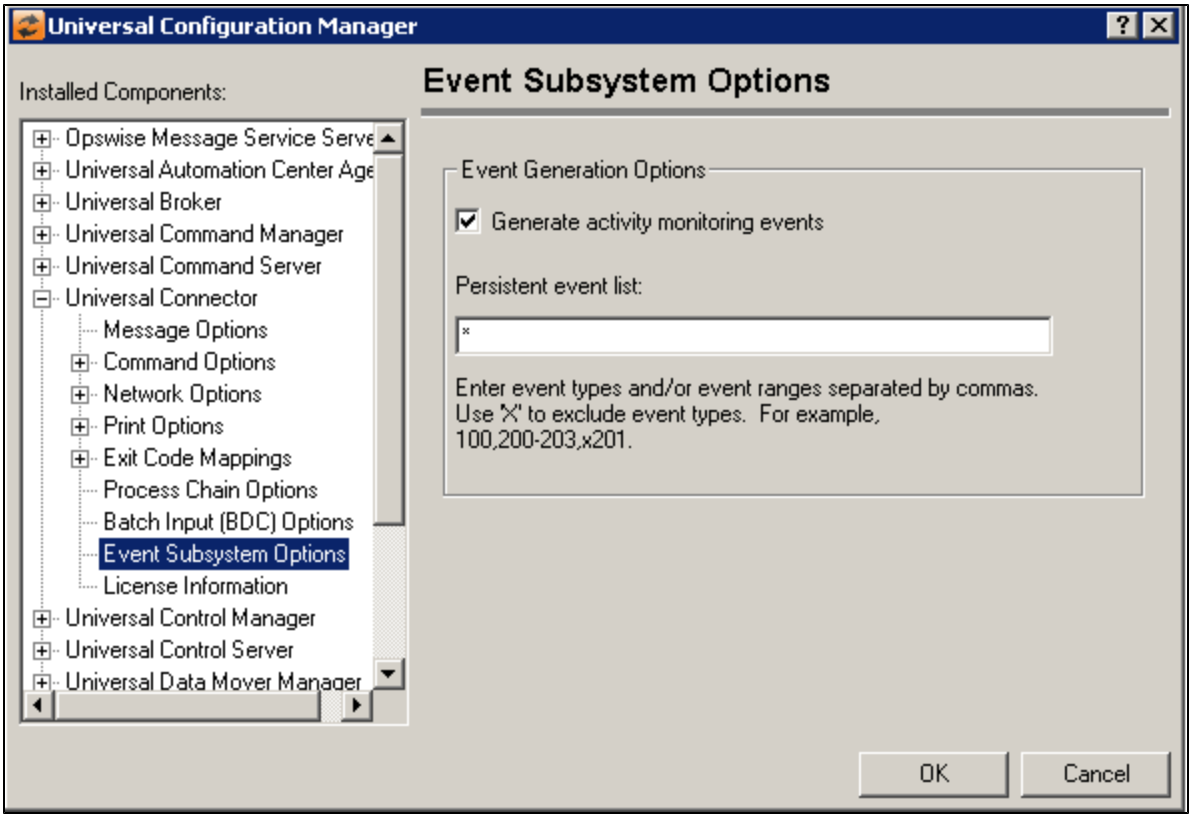
Field	Description
Process Chain Exit Code Mappings	**Green exit code** Specifies the exit code to use for process chains with a status of green. The default value for this option is 0. This option sets the value for the pcgreen_exit_code configuration option. **Red exit code** Specifies the exit code to use for process chains with a status of red. The default value for this option is 4. This option sets the value for the pcred_exit_code configuration option. **Aborted exit code** Specifies the exit code to use for process chains with a status of aborted. The default value for this option is 8. This option sets the value for the pcaborted_exit_code configuration option. **Active exit code** Specifies the exit code to use for process chains with a status of active. The default value for this option is 10. This option sets the value for the pcactive_exit_code configuration option.
Print process chain log	When this option is checked, Universal Connector returns the process chain log to stdout. This option is checked by default. This option sets the value for the print_chainlog configuration option.
Print logs for all processes in process chain	When this option is checked, Universal Connector returns the logs for all jobs in a process chain to stdout. This option is checked by default. This option sets the value for the print_processlogs configuration option.

Universal Connector: Batch Input (BDC) Options



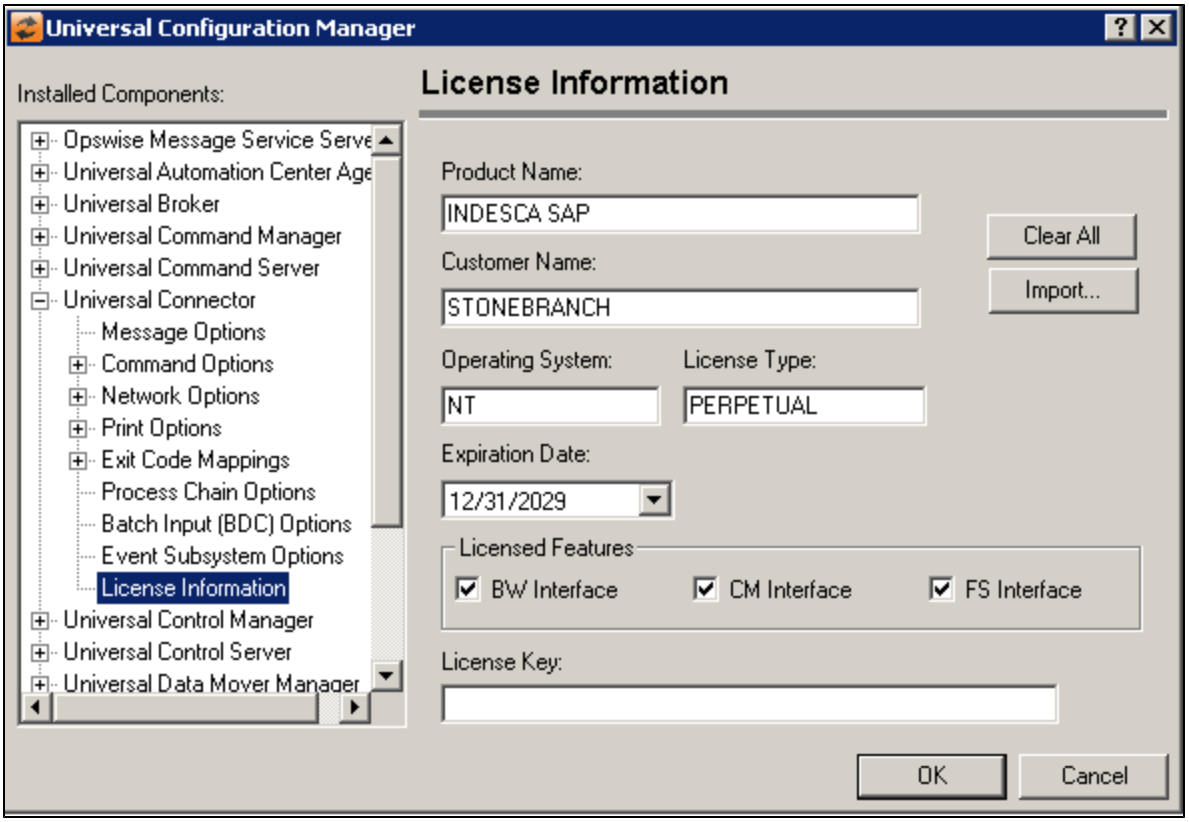
Field	Description
RSBDCSUB Report Search Patterns	**Locate the Job Name using this pattern** Specifies the character pattern that Universal Connector uses to locate the header record and determine the offset of the job name in the RSBDCSUB batch input process report. To reset this option to the default value of Session, delete the field's contents and click OK. **Locate the Job ID using this pattern** Specifies the character pattern that Universal Connector uses to locate the header record and determine the offset of the job ID in the RSBDCSUB batch input process report. To reset this option to the default value of "Job no.", delete the field's contents and click OK. **Locate the Queue ID using this pattern** Specifies the character pattern that Universal Connector uses to locate the header record and determine the offset of the queue ID in the RSBDCSUB batch input process report. To reset this option to the default value of "Queue ID", delete the field's contents and click OK.
Delete batch input session queues that have been successfully processed	When this option is checked, Universal Connector will delete the batch input session queues that have been processed successfully. This option is not selected by default.

Universal Connector: Event Subsystem Options



Field	Description
Event Generation options	**Generate activity monitoring events** When this option is selected, the Universal Connector will generate events that monitor product activity. This option is selected by default. **Persistent event list** Specifies which events are to be generated and processed as persistent events. A persistent event is saved in a Universal Enterprise Controller (UEC) event database for long-term storage. Enter a list and/or range of numeric event types, separated by a comma. To exclude an event or range of events, precede the entry with an upper- or lower-case 'X'. Use an asterisk (*) to specify all event types. For example: • x* Do not generate any events (this is the default) • * Generate all event types • 100 Generate all event types except 100 • 100,200-205 Generate event type 100 and 200 through 205 • 100,200-205,x202 Generate event types 100 and 200 thru 205, excluding event type 202 Entries are processed from left to right, and the list is order dependent. For example, an entry of x*,100 prevents generation of all event types except 100, while an entry of 100,x* prevents generation of all event types (the right-most entry takes precedence).

Universal Connector: License Information



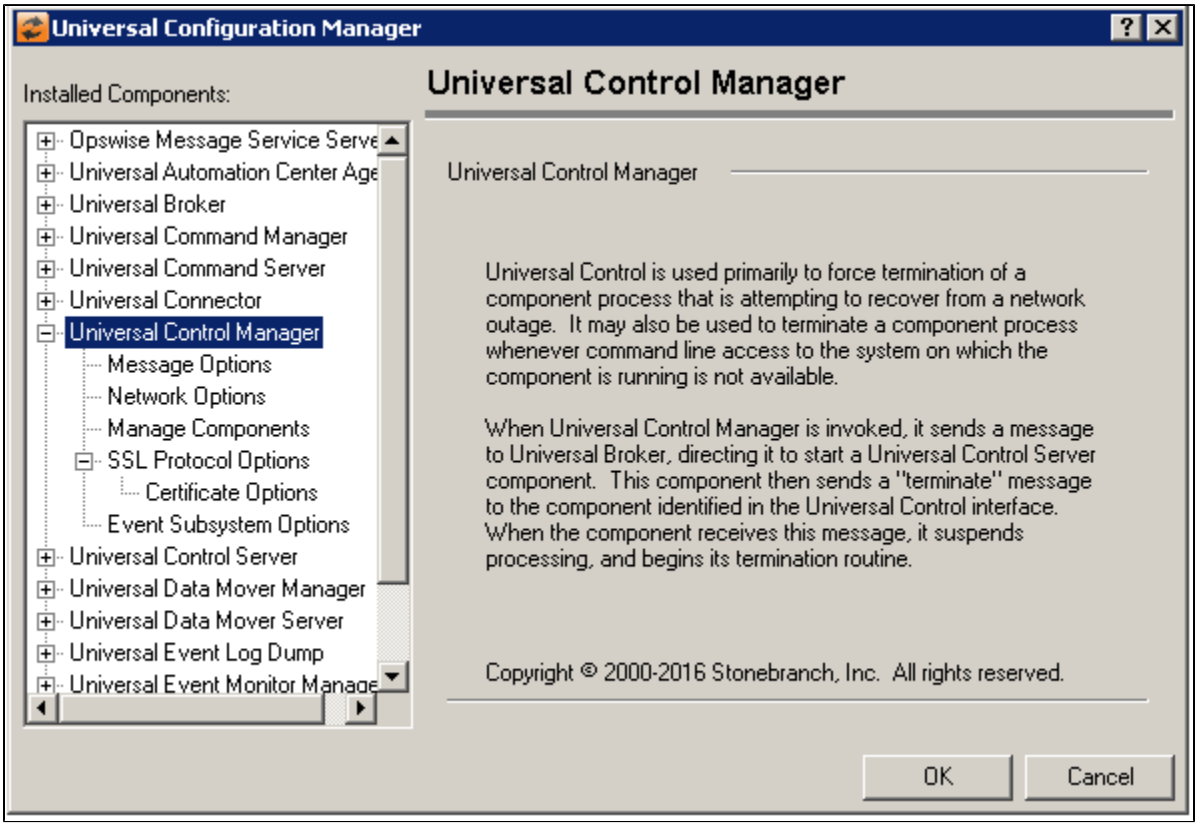
Field	Description
Product Name	Specifies the name of the registered product for which the license key was issued.
Customer Name	Contains the name under which the software is registered. This value must be entered exactly as the customer name specified in the licensing material you received with the software.
Operating System	Identifies the operating system on which your licensed Universal Command Manager will run. This value should match the operating system specified in the licensing material you received with your software.
License Type	Specifies the type of license issued for the current product. This value is used with the expiration date to determine the length of time the software license is valid.
Expiration Date	Contains the date when the current license for this product will expire. The date entered must match the expiration date specified in the licensing material you received with the software. To change the date, enter a new value, or click the down arrow to display a calendar control.
Server Counts	Contains the number of Universal Command Servers you have licensed, by operating system. This values in this list should match the numbers specified in the licensing material you received with your software.
License Key	Contains the license key that is generated for each product, customer, license type and expiration date. The value contained in this field must match the key specified in the licensing material you received with the software package.
Clear all Button	Select this button to reset the license information fields to their initialized defaults.
Import Button	Select this button to browse for and load the Universal Command Manager license file, which will automatically import the license information.

Universal Configuration Manager - Universal Control Manager

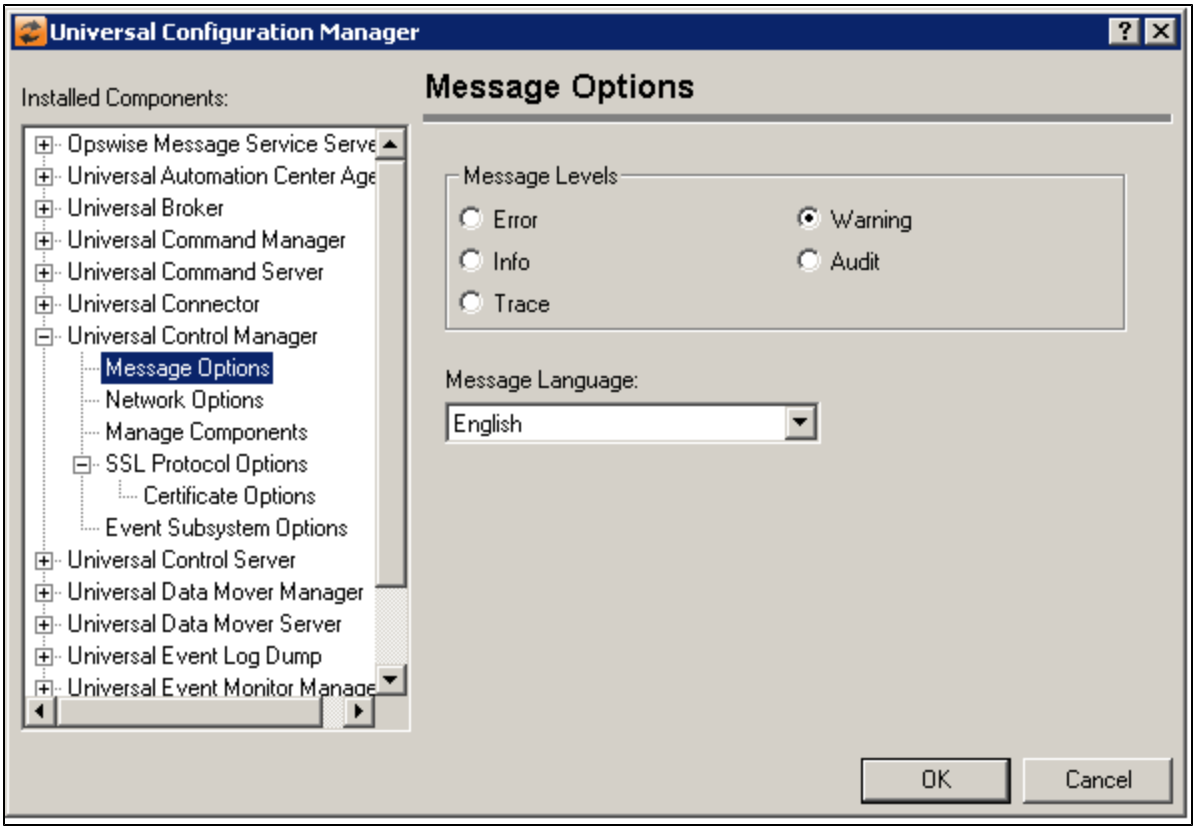
- Universal Control Manager
- Universal Control Manager: Message Options
- Universal Control Manager: Network Options
- Universal Control Manager: Manage Components

- Universal Control Manager: SSL/TLS Protocol Options
  - Universal Control Manager: SSL/TLS Protocol Options - Certificate Options
- Universal Control Manager: Event Subsystem Options

Universal Control Manager

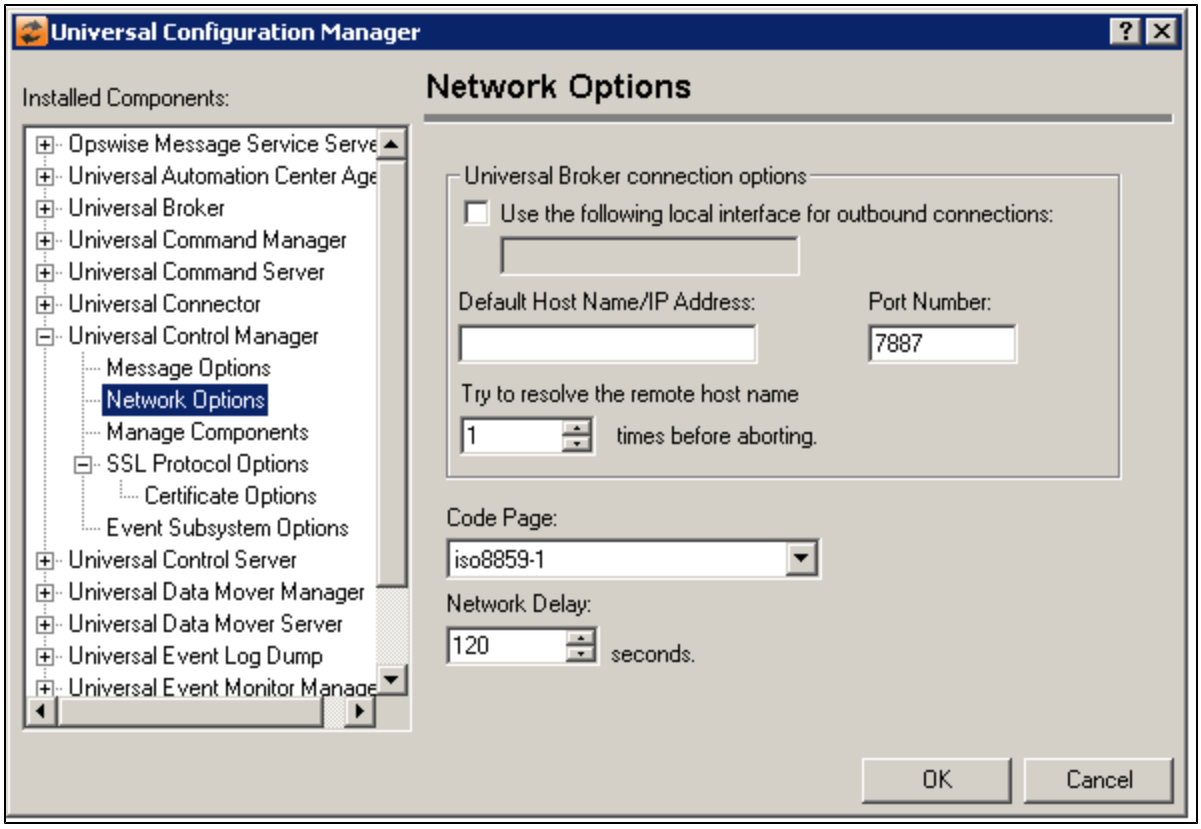


Universal Control Manager: Message Options



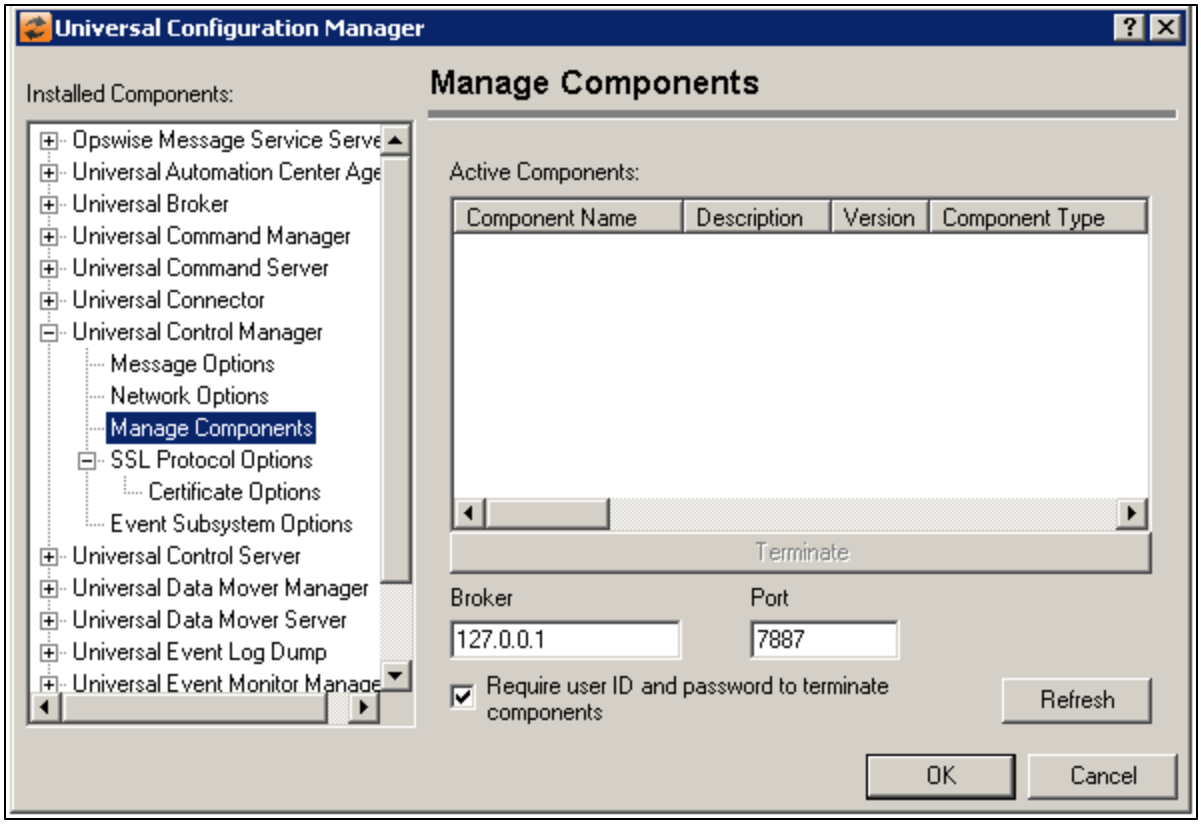
Field	Description
Message Level	**Error** Turns on message logging for errors only. **Warning** Turns on message logging for warnings and errors. **Info** Turns on message logging for all informational, warning and error messages. **Audit** Turns on message logging for all audit, informational, warning and error messages. **Trace** Allows all messages to be logged, including those used for debugging purposes.
Message Language	Specifies the language used to display error messages.

Universal Control Manager: Network Options



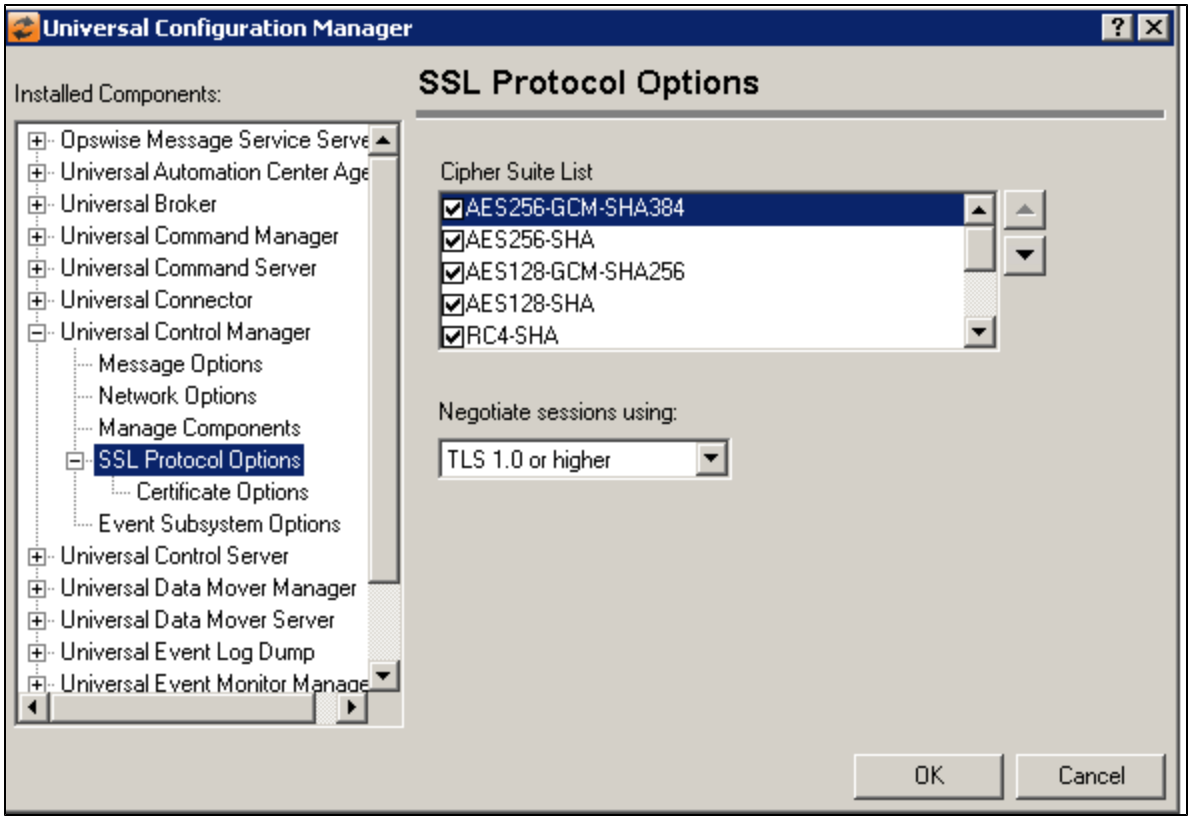
Field	Description
Universal Broker Connection Options	**Use the following local interface for outbound connections** When this option is selected, the IP address or alias of a local interface may be specified that will be used whenever UCtl Manager connects to a Universal Broker. If this option is not selected, any available local interface will be used. **Default Host Name/IP Address** Enter a valid IP address or host name that will be used as the default system for process execution. To run processes on the local system, enter a host name of localhost, or an IP address of 127.0.0.1. **Port Number** Enter a port number or service name that will be used to connect to Universal Broker. **Try to resolve the remote host name** Indicates the number of attempts that will be made to resolve a host name specified with the host command/configuration option. This parameter can help resolve intermittent "host not found" errors.
Code Page	Sets the translation table used to translate text-based data transmitted across the network.
Network Delay	The network delay value specifies the maximum acceptable delay in the network. Should it take longer then the specified delay time to send or receive data over the network, the operation will end with a time out error. The default is 120 seconds.

Universal Control Manager: Manage Components

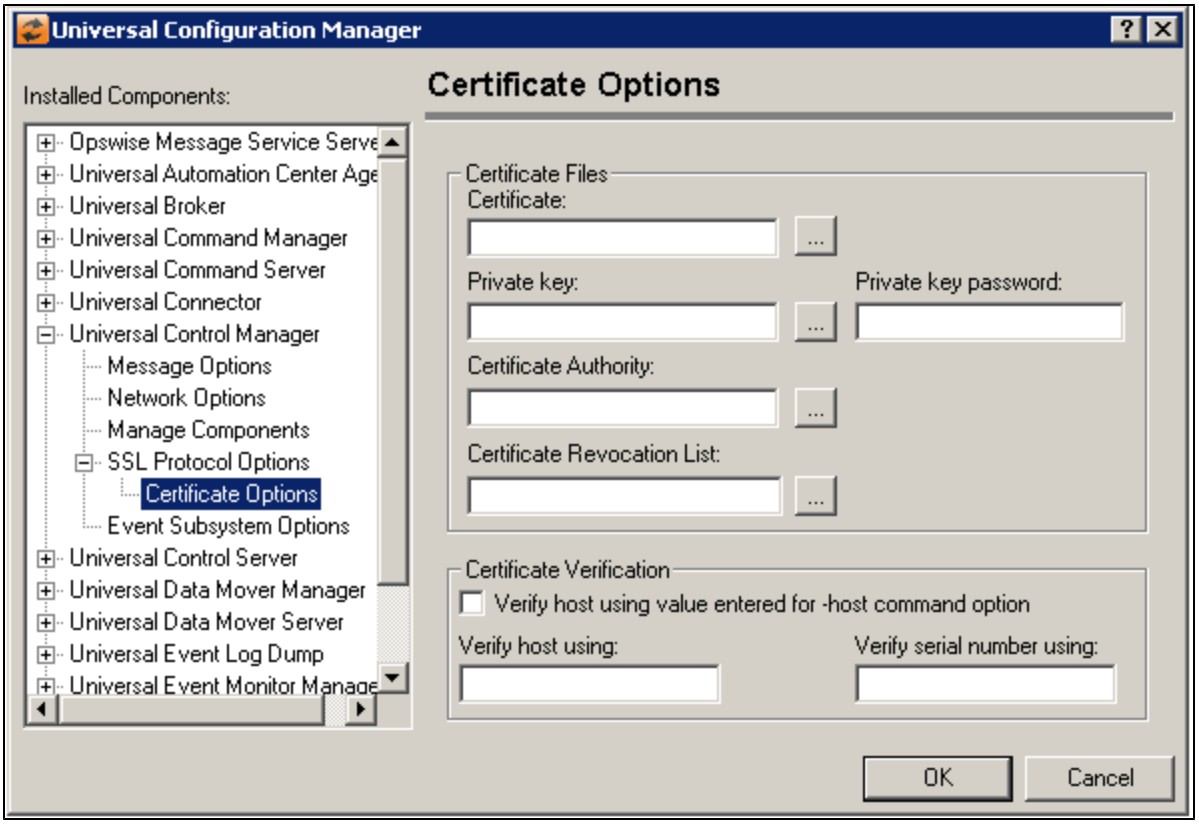


Field	Description
Active Components	Contains a list of components that are currently running.
Broker	Address of the Broker to query.
Port	Port the Broker being queried is listening on.
Require user ID and password to terminate components	When checked, a dialog will be displayed when the Terminate button is selected that will prompt for a user ID and password. This is necessary if either of the following are true: a) the Universal Control Server is configured to run with security, or b) the Universal Control Server has an ACL entry that requires user authentication. • If the Universal Control Server is configured to run without security, this option may be left unchecked. • If this option is not checked and the terminate request fails, set this option and resubmit the terminate request, supplying a valid userid and password.
Refresh Button	Updates the list of active components.

Universal Control Manager: SSL/TLS Protocol Options

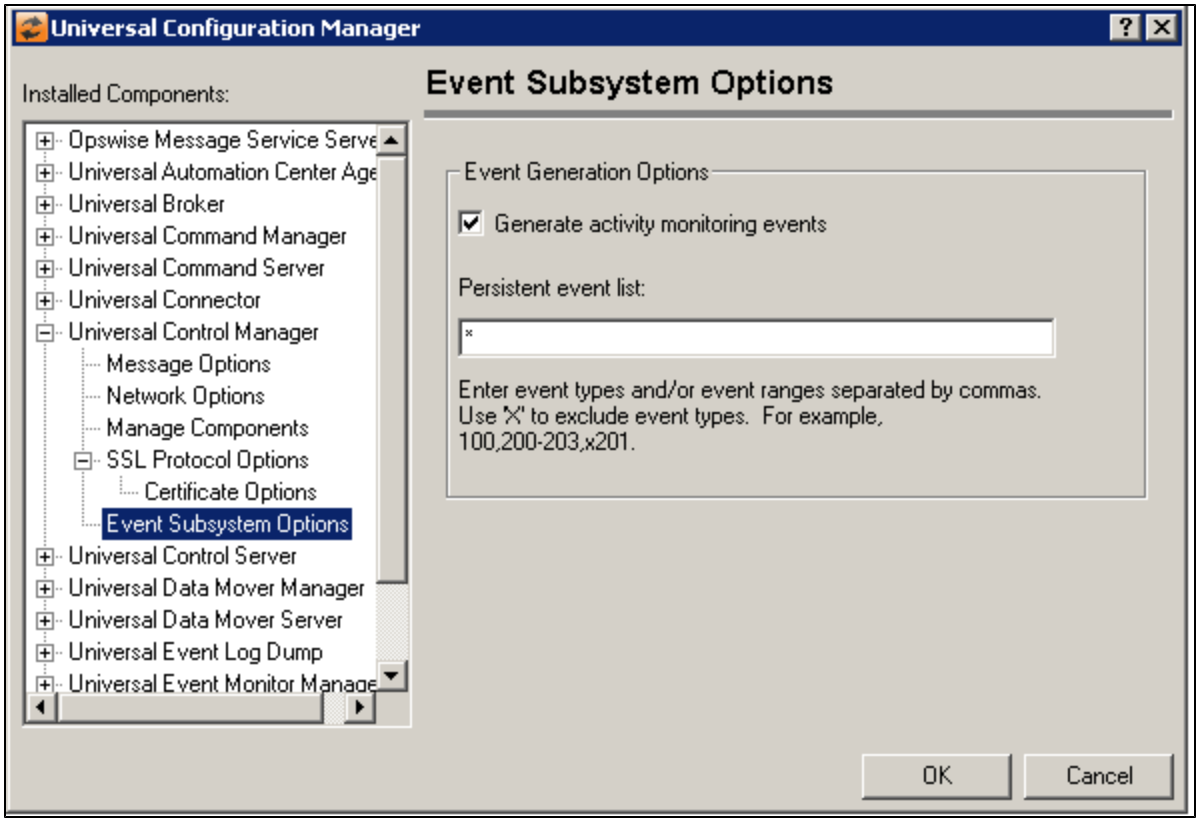


Field	Description
Cipher Suite List	The cipher suite specifies the symmetric and message digest algorithms used for a session. The cipher suite is negotiated as part of the SSL/TLS handshake. If one of the cipher suites in the list cannot be agreed upon, the SSL/TLS handshake will fail, and the connection will be closed.
Negotiate Sessions using:	(No help provided.)



Field	Description
Certificate Files	**Certificate** The Certificate option specifies the X.509 certificate file that identifies the program to remote components with which it connects using the SSL/TLS protocol. **Private Key** The Private key option specifies the private key file associated with the X.509 certificate specified in the Certificate file. **Private Key Password** The Private key password specifies the password associated with the private key file. **Certificate Authority** The Certificate Authority option specifies a file with a list of trusted authorities. Any certificate received that is signed by a trusted Certificate Authority is accepted. **Certificate Revocation List** The Certificate Revocation List option specifies a file with a list of revoked certificates. Any certificate received that is in this list will not be accepted.
Certificate Verification	**Verify host using value entered for -host command option** The host name specifies that the ubroker certificate's host name value must match either the -host option value or the value specified by the entered host name. There is no default. **Verify host Using** The host name specifies that the ubroker certificate's host name value must match either the -host option value or the value specified by the entered host name. There is no default. **Verify serial number using** The serial number specifies that the ubroker's X.509 certificate serial number is verified to match the value specified. The number is considered to be in decimal format unless it is prefixed with 0x or 0X in which case it is considered a hexidecimal format. There is no default.

Universal Control Manager: Event Subsystem Options



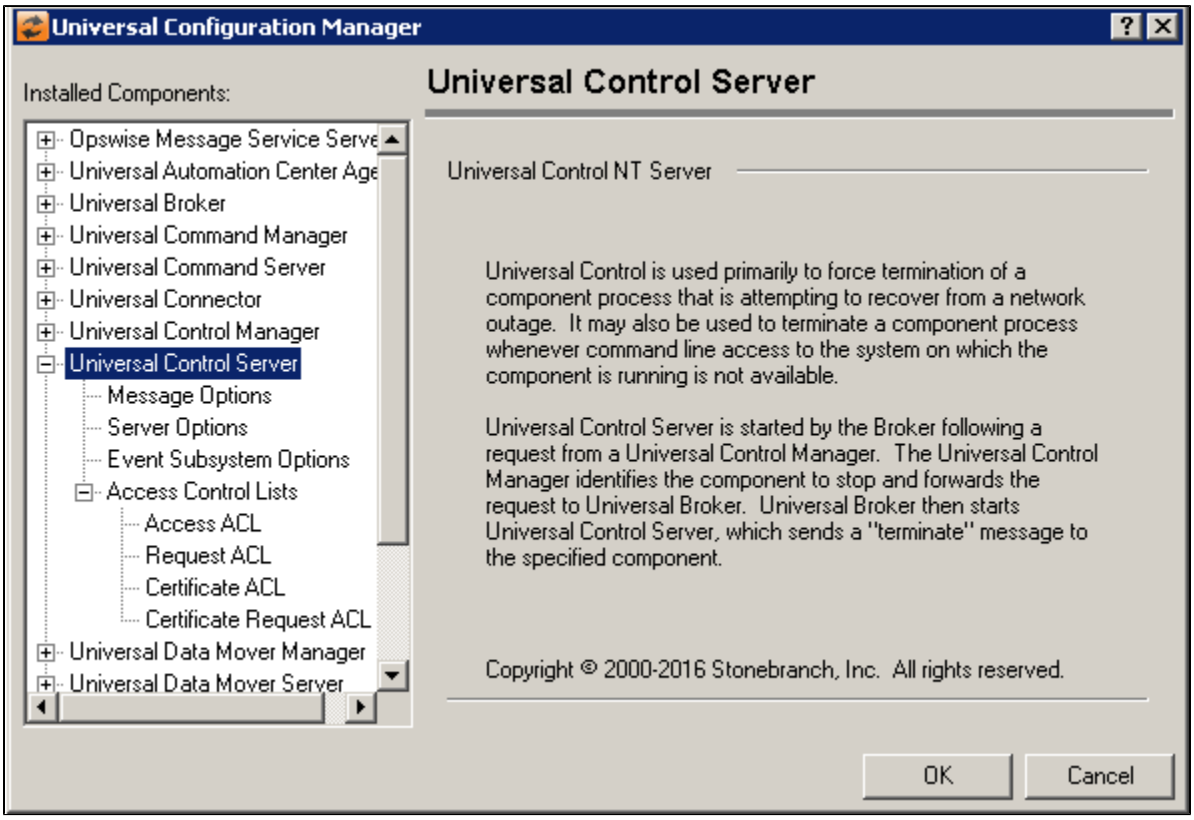
Field	Description
Event Generation Options	**Generate activity monitoring events** When this option is selected, the Universal Control Manager will generate events that monitor product activity. This option is selected by default. **Persistent event list** Specifies which events are to be generated and processed as persistent events. A persistent event is saved in a Universal Enterprise Controller (UEC) event database for long-term storage. Enter a list and/or range of numeric event types, separated by a comma. To exclude an event or range of events, precede the entry with an upper- or lower-case 'X'. Use an asterisk (*) to specify all event types. For example: • x* Do not generate any events (this is the default) • * Generate all event types • 100 Generate all event types except 100 • 100,200-205 Generate event type 100 and 200 through 205 • 100,200-205,x202 Generate event types 100 and 200 through 205, excluding event type 202 Entries are processed from left to right, and the list is order dependent. For example, an entry of x*,100 prevents generation of all event types except 100, while an entry of 100,x* prevents generation of all event types (the right-most entry takes precedence).

Universal Configuration Manager - Universal Control Server

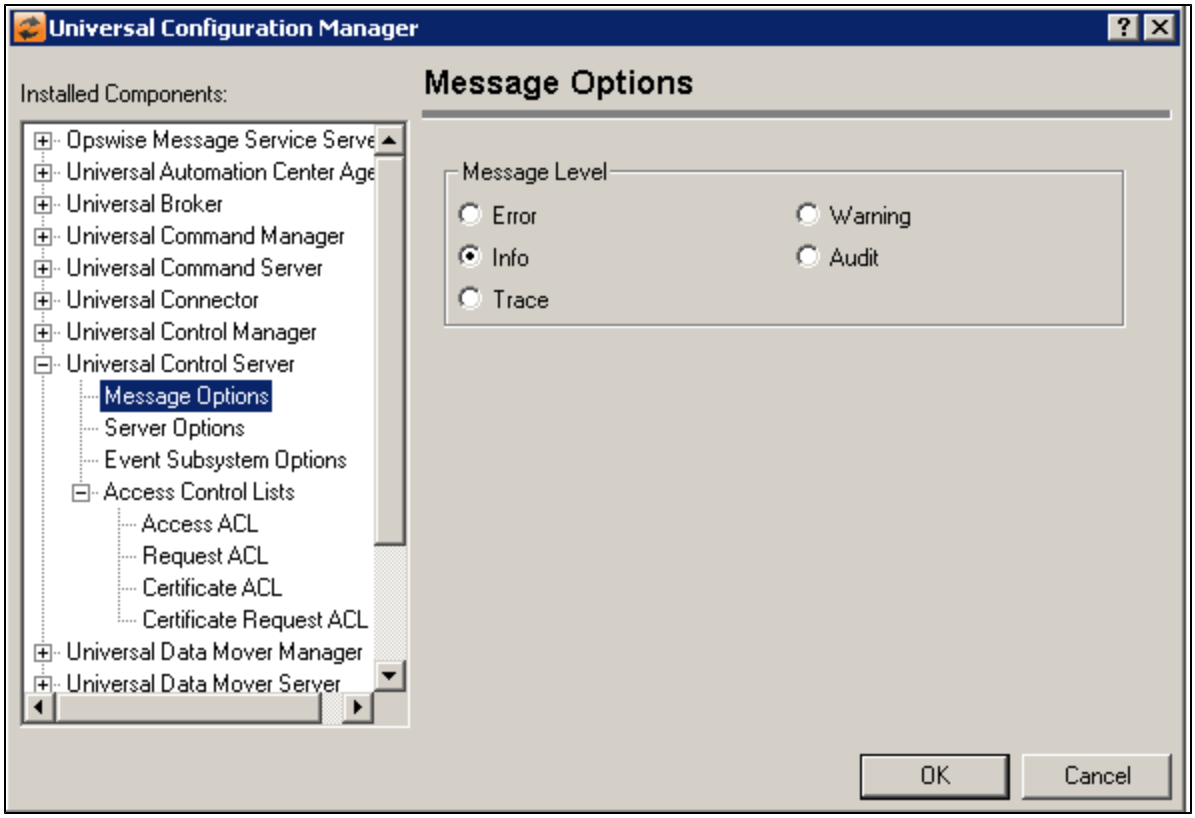
- Universal Control Server
- Universal Control Server: Message Options
- Universal Control Server: Server Options
- Universal Control Server: Event Subsystem Options
- Universal Control Server: Access Control Lists
  - Universal Control Server: Access Control Lists - Access ACL

- Universal Control Server: Access Control Lists - Request ACL
- Universal Control Server: Access Control Lists - Certificate ACL
- Universal Control Server: Access Control Lists - Certificate Request ACL

Universal Control Server

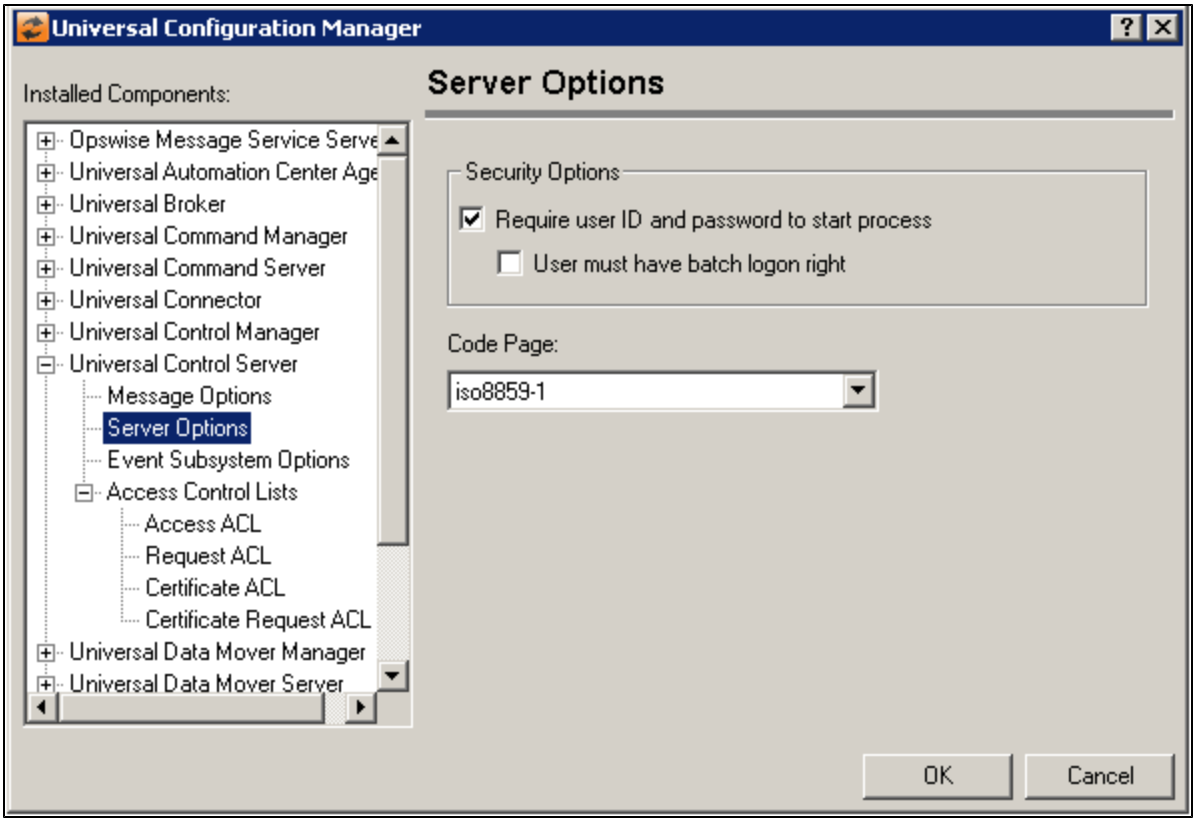


Universal Control Server: Message Options



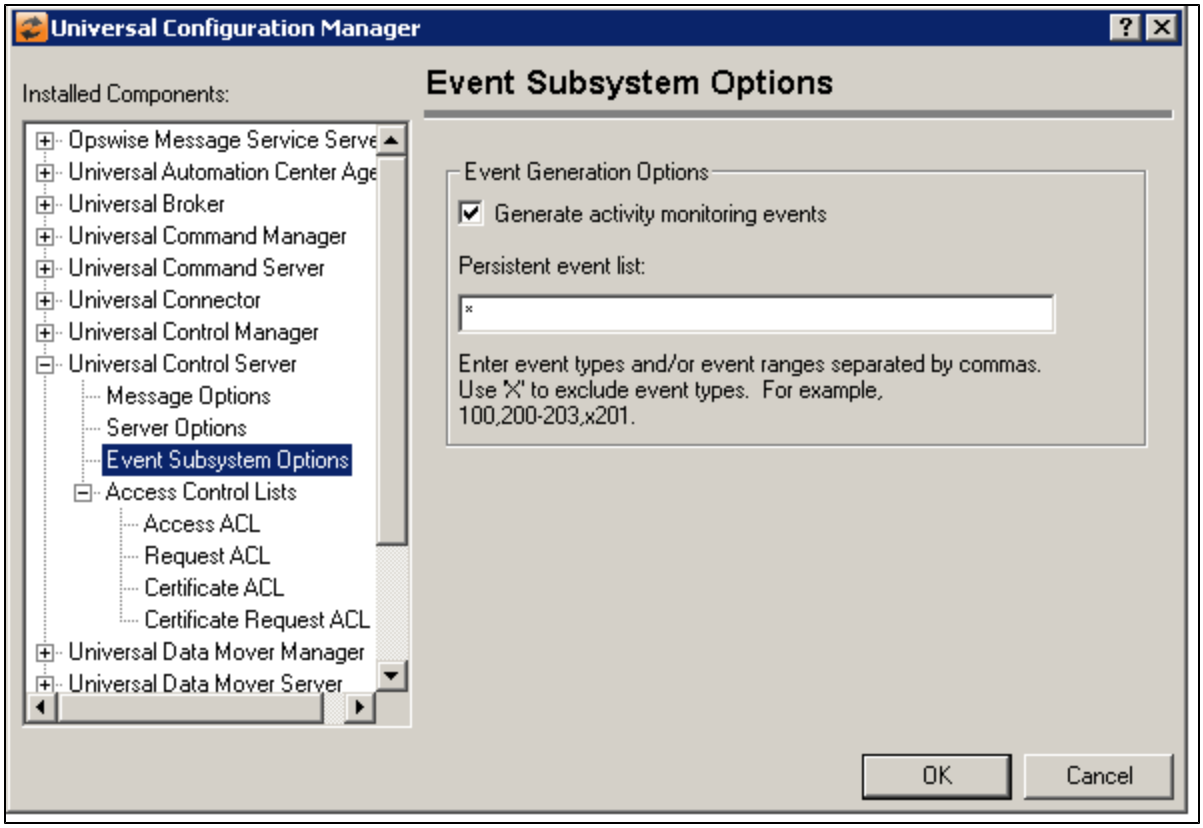
Field	Description
Message Level	**Error** Turns on message logging for errors only. **Warning** Turns on message logging for warnings and errors. **Info** Turns on message logging for all informational, warning and error messages. **Audit** Turns on message logging for all audit, informational, warning and error messages. **Trace** Allows all messages to be logged, including those used for debugging purposes.

Universal Control Server: Server Options



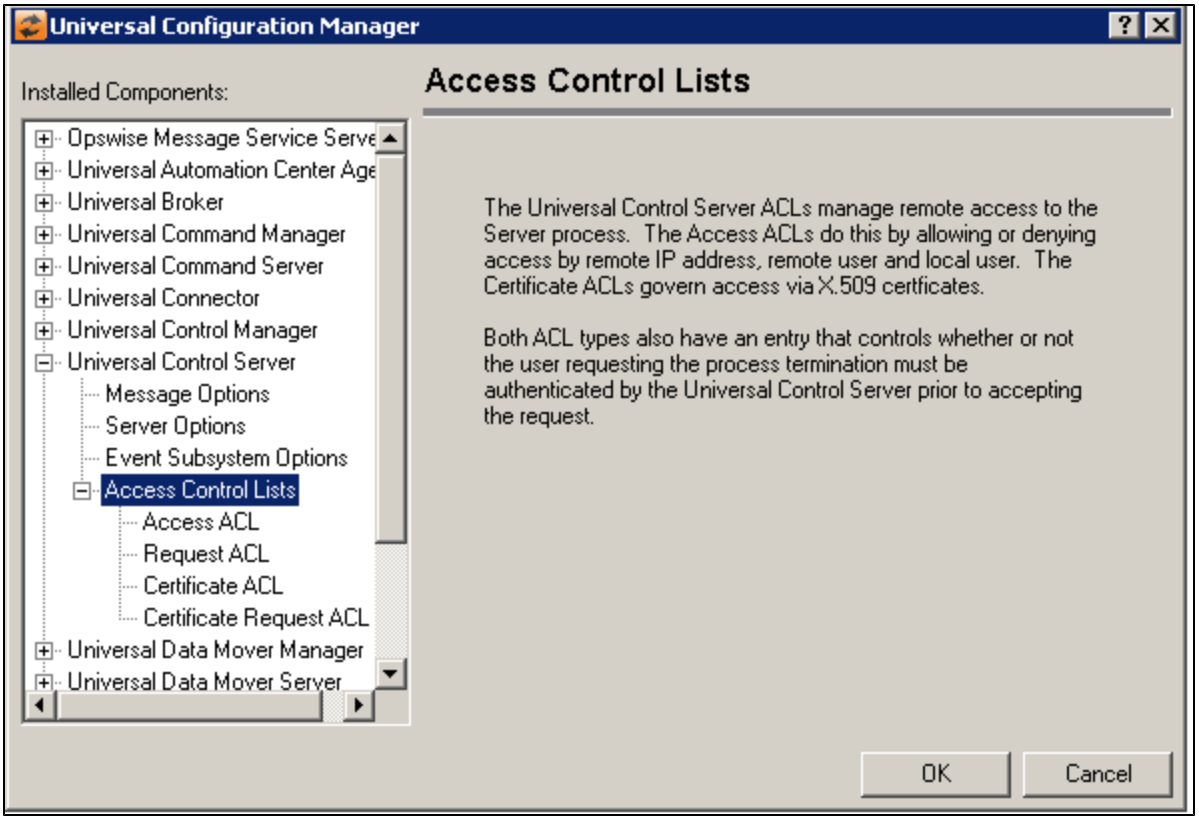
Field	Description
Security Options	**Require User ID and Password to start process** When this option is checked, Universal Command will require a valid user ID and password to start a process. **User must have batch logon right** When this option is checked, the Log on as a batch job user right must be granted to the user ID passed to Universal Command before the process will run. This is an advanced user right that must be set using the NT User Manager.
Code Page	Sets the translation table used to translate text-based data transmitted across the network.

Universal Control Server: Event Subsystem Options

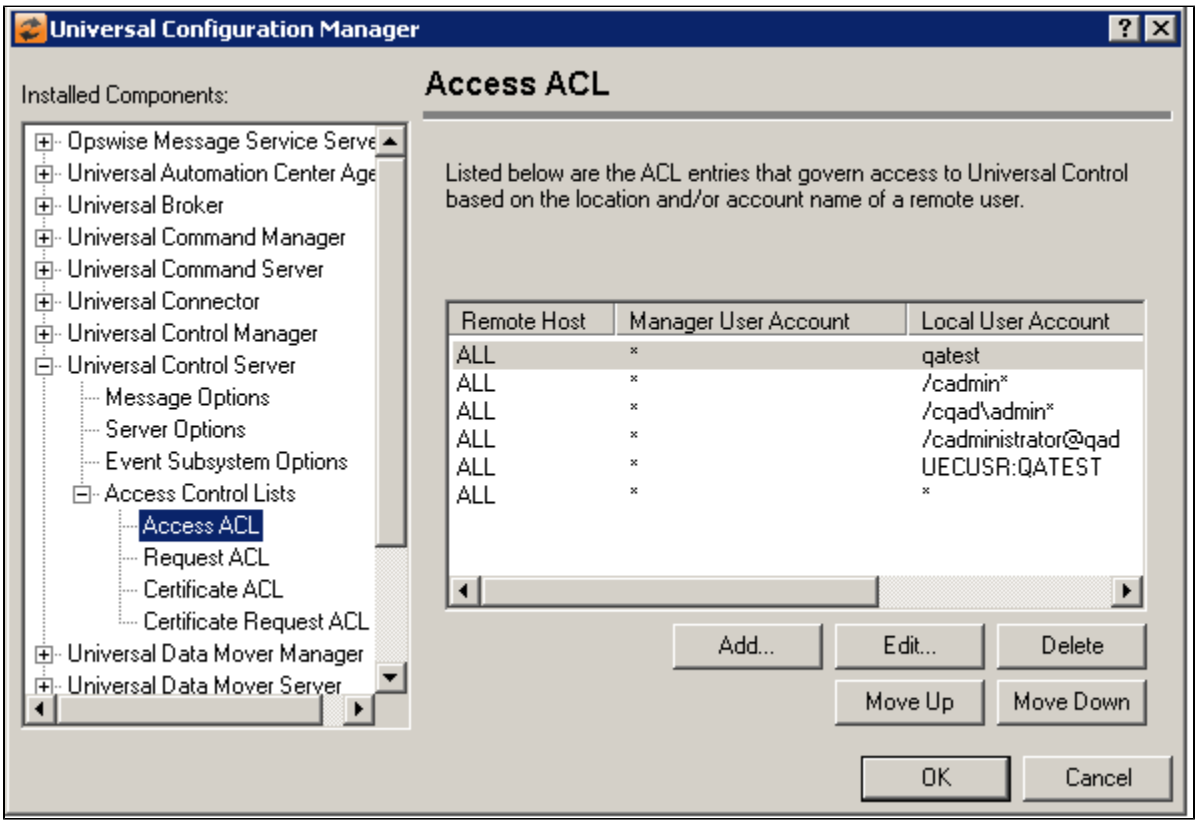


Field	Description
Event Generation Options	**Generate activity monitoring events** When this option is selected, the Universal Control Server will generate events that monitor product activity. This option is selected by default. **Persistent event list** Specifies which events are to be generated and processed as persistent events. A persistent event is saved in a Universal Enterprise Controller (UEC) event database for long-term storage. Enter a list and/or range of numeric event types, separated by a comma. To exclude an event or range of events, precede the entry with an upper- or lower-case 'X'. Use an asterisk to specify all event types. For example: • x* Do not generate any events (this is the default) • * Generate all event types • 100 Generate all event types except 100 • 100,200-205 Generate event type 100 and 200 through 205 • 100,200-205,x202 Generate event types 100 and 200 thru 205, excluding event type 202 Entries are processed from left to right, and the list is order dependent. For example, an entry of x*,100 prevents generation of all event types except 100, while an entry of 100,x* prevents generation of all event types (the right-most entry takes precedence).

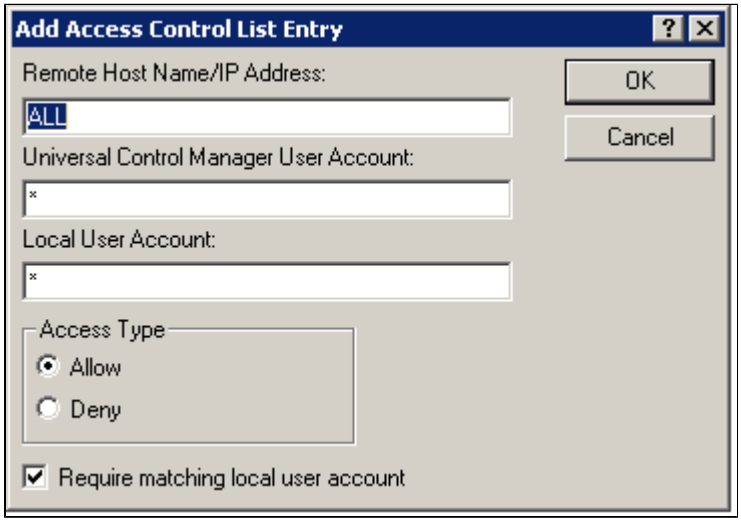
Universal Control Server: Access Control Lists

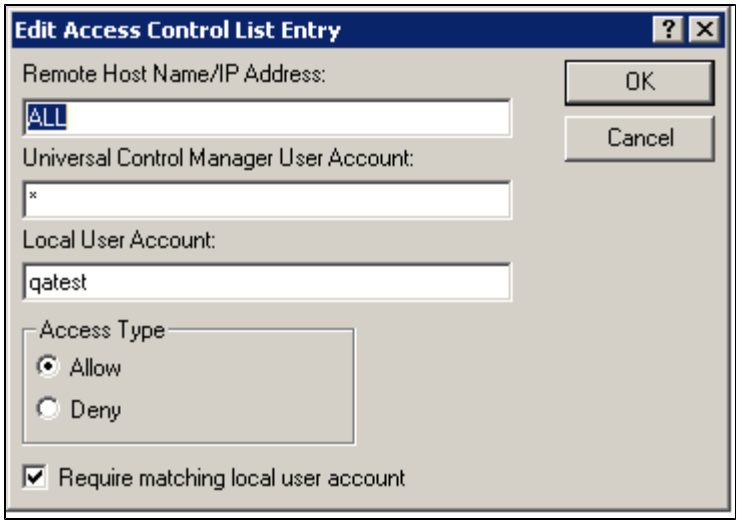


Universal Control Server: Access Control Lists - Access ACL



Field	Description

Access Control List	Displays the entries contained in the Universal Control Server Access Control List (ACL). This list is used to grant or deny access to Universal Control Server based on the remote system's IP address, the user account running the Universal Control Manager, and the user account with which the Server is executed. The order in which ACL entries are evaluated and applied are based on their position in this list. Use the Move Up and Move Down buttons to change an entry's priority.
Add Button	Displays a dialog box that allows an entry to be added to the Access Control List. The fields in this dialog are populated with default values.  **Remote Host Name/IP Address** The IP address or name of the remote host for which access will be allowed or denied. To apply an access control entry to all IP addresses, use the keyword ALL (in all caps). **Universal Control Manager User Account** The ID of the user account executing the Universal Control Manager. **Local User Account** A local user account that may be provided to UCTL Server for authentication. **Access Type** Will cause the Universal Control Server to accept/reject incoming connections from the specified host, provided the Universal Control Manager is executed by the specified user account and the process is owned by the specified local user account. **Requires Matching local user account** When this option is checked, a local user account must be supplied to the Universal Control Server that matches the user account in whose security context the Universal Server component is running. This option only applies to Stop requests, and is effectively disabled for Refresh and Start requests. To prevent unauthorized users from issuing those requests, make sure security is enabled in the UCtl Server's configuration.
Edit Button	Displays a dialog that allows the selected Access Control List entry to be modified.



Remote Host Name/IP Address
The IP address or name of the remote host for which access will be allowed or denied. To apply an access control entry to all IP addresses, use the keyword ALL (in all caps).

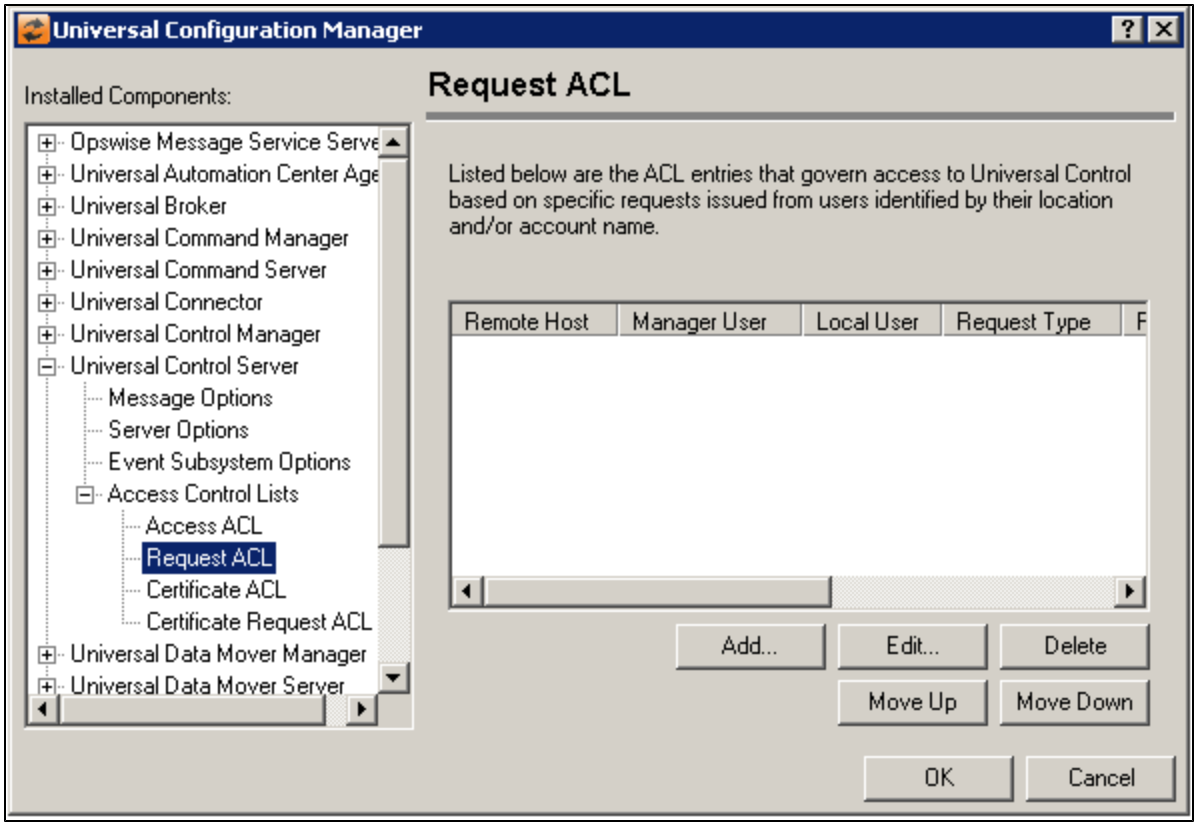
Universal Control Manager User Account
The ID of the user account executing the Universal Control Manager.

Local User Account
A local user account that may be provided to UCTL Server for authentication.

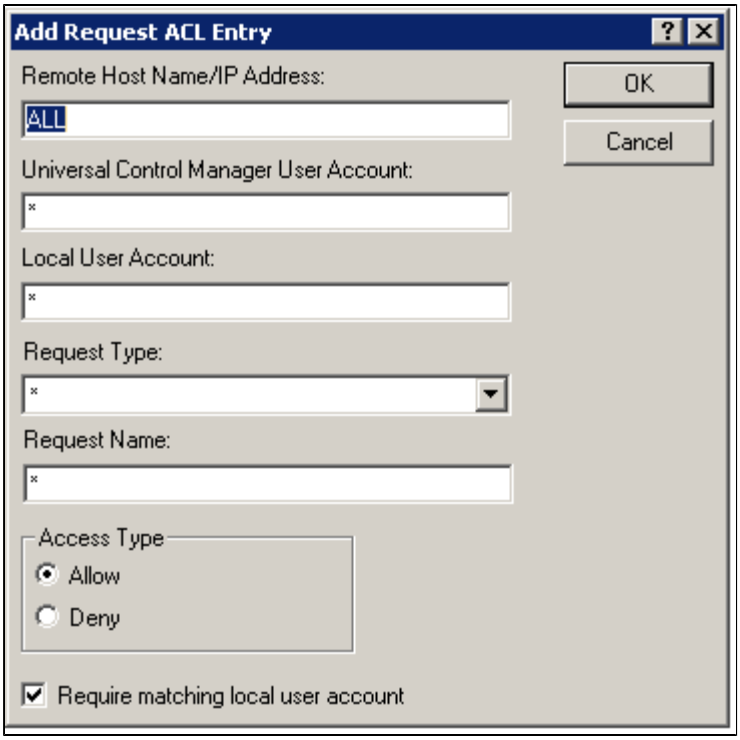
Access Type
Will cause the Universal Control Server to accept/reject incoming connections from the specified host, provided the Universal Control Manager is executed by the specified user account and the process is owned by the specified local user account.

Requires Matching local user account
When this option is checked, a local user account must be supplied to the Universal Control Server that matches the user account in whose security context the Universal Server component is running. This option only applies to Stop requests and is effectively disabled for Refresh and Start requests. To prevent unauthorized users from issuing those requests, make sure security is enabled in the UCtl Server's configuration.

Delete Button	Removes the selected item from the Access Control List.
Move Up Button	Moves the selected item up one position in the list.
Move Down Button	Moves the selected item down one position in the list.



Field	Description
Request ACL List	Displays the entries contained in the Universal Control Server Access Control List (ACL). This list is used to grant or deny the ability to issue specific Universal Control Server requests based on the remote system's IP address, the user account running the Universal Control Manager, and a locally authenticated user account. The order in which ACL entries are evaluated and applied are based on their position in this list. Use the Move Up and Move Down buttons to change an entry's priority.
Add Button	Displays a dialog box that allows an entry to be added to the Access Control List. The fields in this dialog are populated with default values.



Remote Host Name/IP Address
The IP address or name of the remote host for which access will be allowed or denied. To apply an access control entry to all IP addresses, use the keyword ALL (in all caps).

Universal Command Manager User Account
The ID of the user account executing the Universal Control Manager.

Local User Account
A local user account that may be provided to UCTL Server for authentication.

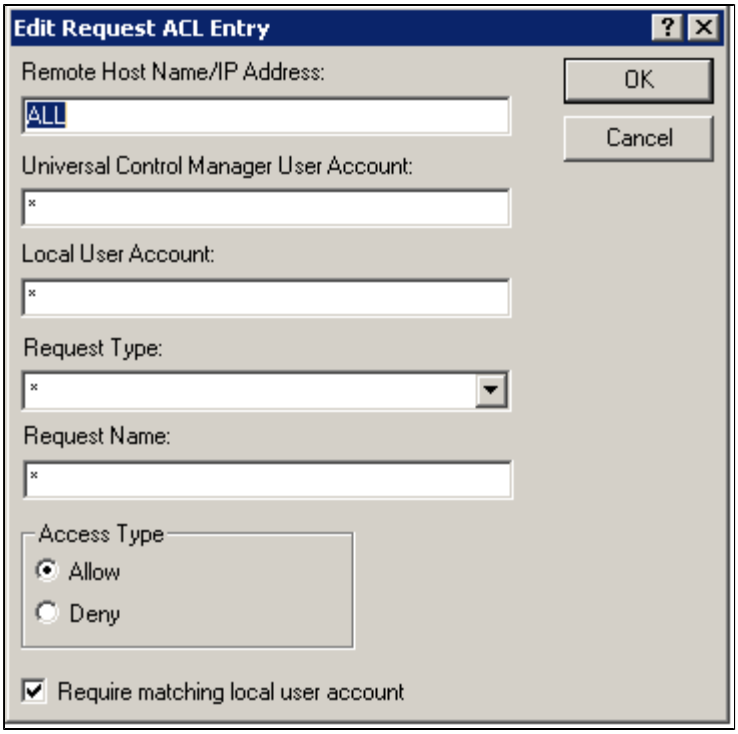
Request Type
Specifies the type of request issued by the UCTL Manager. Valid request types are included in the list. * may also be specified to indicate the rule applies to all valid UCTL Manager request types.

Request Name
Enter the name of a Universal component here to set access for the specified local user account based on the UCTL Manager request. Use the default value of * to grant or deny access to all requests issued by the user.

Access Type
Will cause the Universal Control Server to accept/reject incoming connections from the specified host, provided the Universal Control Manager is executed by the specified user account and the process is owned by the specified local user account.

Requires Matching local user account
When this option is checked, a local user account must be supplied to the Universal Control Server that matches the user account in whose security context the Universal Server component is running. This option only applies to Stop requests and is effectively disabled for Refresh and Start requests. To prevent unauthorized users from issuing those requests, make sure security is enabled in the UCtl Server's configuration.

Edit Button	Displays a dialog that allows the selected Access Control List entry to be modified.



Remote Host Name/IP Address
The IP address or name of the remote host for which access will be allowed or denied. To apply an access control entry to all IP addresses, use the keyword ALL (in all caps).

Universal Control Manager User Account
The ID of the user account executing the Universal Control Manager.

Local User Account
A local user account that may be provided to UCTL Server for authentication.

Request Type
Specifies the type of request issued by the UCTL Manager. Valid request types for UCmd Servers running on Windows are included in the list. * may also be specified to indicate the rule applies to all valid UCmd Manager request types.

Request Name
Enter the name of a Universal component here to set access for the specified local user account based on the UCTL Manager request. Use the default value of * to grant or deny access to all requests issued by the user.

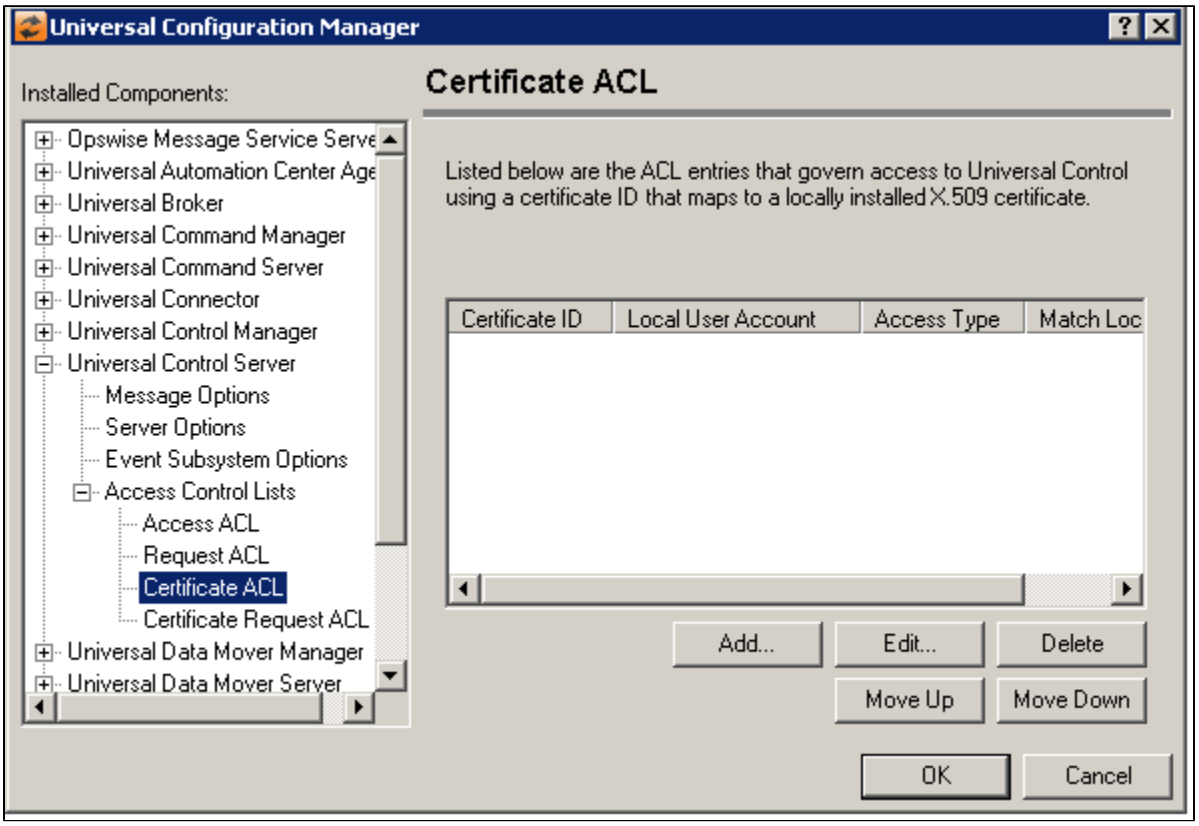
Access Type
Will cause the Universal Control Server to accept/reject incoming connections from the specified host, provided the Universal Control Manager is executed by the specified user account and the process is owned by the specified local user account.

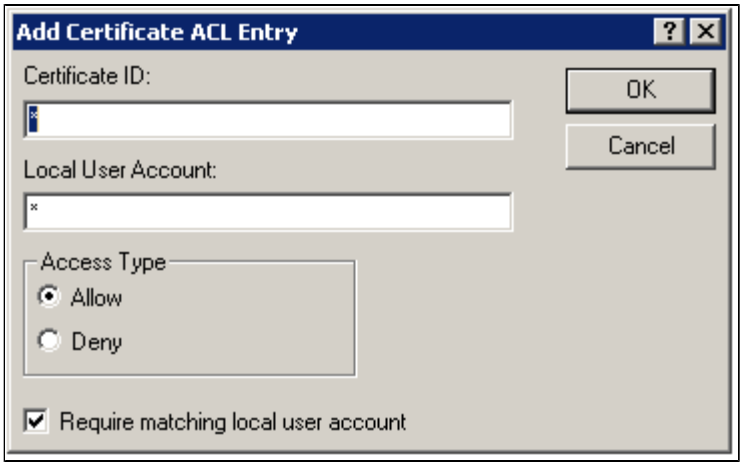
Requires Matching local user account
When this option is checked, a local user account must be supplied to the Universal Control Server that matches the user account in whose security context the Universal Server component is running. This option only applies to Stop requests and is effectively disabled for Refresh and Start requests. To prevent unauthorized users from issuing those requests, make sure security is enabled in the UCtl Server's configuration.

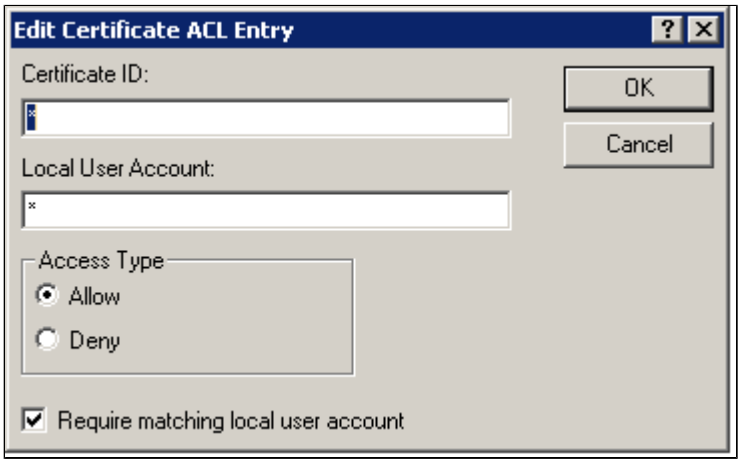
Delete Button	Removes the selected item from the Access Control List.
Move Up Button	Moves the selected item up one position in the list.
Move Down	Moves the selected item down one position in the list.

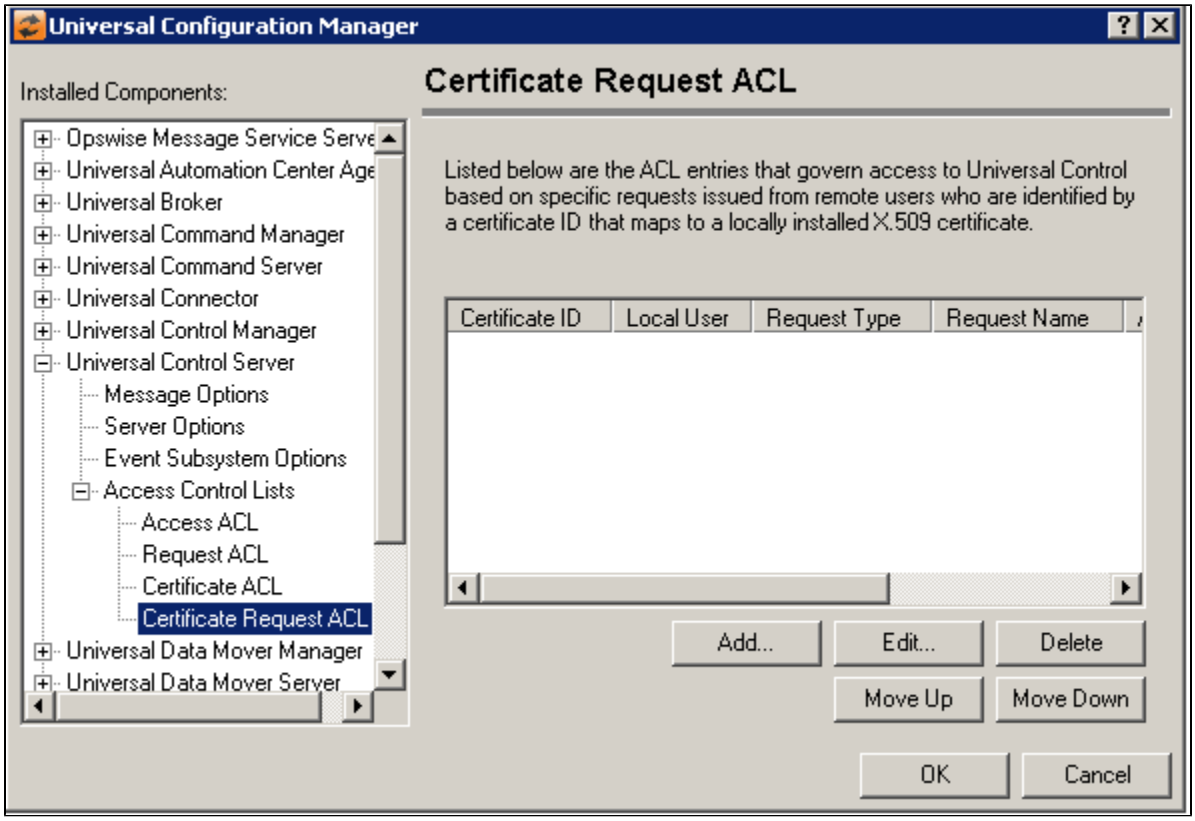
But ton	

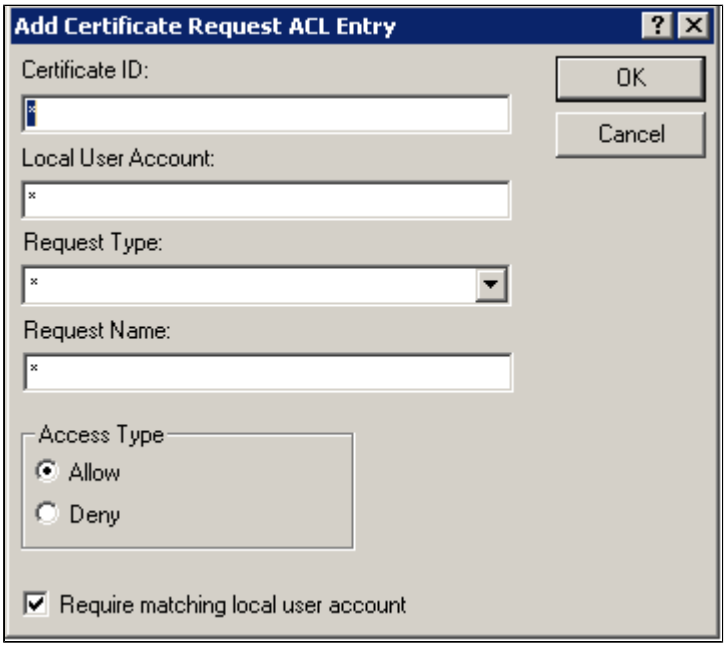
Universal Control Server: Access Control Lists - Certificate ACL



Field	Description
Certificate ACL List	Displays the entries contained in the Universal Control Certificate Access Control List (ACL). The order in which ACL entries are evaluated and applied are based on their position in this list. Use the Move Up and Move Down buttons to change an entry's priority.
Add Button	Displays a dialog box that allows an entry to be added to the Access Control List. The fields in this dialog are populated with default values.  **Certificate ID** When a UCTL Manager specifies a certificate ID from its command line, its corresponding UCTL Server compares it to the certificate ID entered here. If a match is found and the certificate ID has a corresponding entry in the Certificate Map list, the authority to execute the

	request is granted or denied to the specified local user account. **Local User Account** A local user account that may be provided to UCTL Server for authentication. **Access Type** Will cause the Universal Control Server to accept/reject incoming connections from the specified host, provided the Universal Control Manager is executed by the specified user account and the process is owned by the specified local user account. **Requires Matching local user account** When this option is checked, a local user account must be supplied to the Universal Control Server that matches the user account in whose security context the Universal Server component is running. This option only applies to Stop requests and is effectively disabled for Refresh and Start requests. To prevent unauthorized users from issuing those requests, make sure security is enabled in the UCtl Server's configuration.
Edit Button	Displays a dialog that allows the selected Access Control List entry to be modified.  **Certificate ID** When a Universal Control Manager specifies a certificate ID from its command line, its corresponding UCTL Server compares it to the certificate ID entered here. If a match is found and the certificate ID has a corresponding entry in the Certificate Map list, the authority to execute the request is granted or denied to the specified local user account. **Local User Account** A local user account that may be provided to UCTL Server for authentication. **Access Type** Will cause the Universal Control Server to accept/Reject incoming connections from the specified host, provided the Universal Control Manager is executed by the specified user account and the process is owned by the specified local user account. **Requires Matching local user account** When this option is checked, a local user account must be supplied to the Universal Control Server that matches the user account in whose security context the Universal Server component is running. This option only applies to Stop requests and is effectively disabled for Refresh and Start requests. To prevent unauthorized users from issuing those requests, make sure security is enabled in the UCtl Server's configuration.
Delete Button	Removes the selected item from the Access Control List.
Move Up Button	Moves the selected item up one position in the list.
Move Down Button	Moves the selected item down one position in the list.



Field	Description
Certificate Request ACL List	Displays the entries contained in the Universal Control Server Access Control List (ACL). This list is used to grant or deny access to specific Universal Control Server requests based on a remote user who is identified by a certificate ID (which maps to a locally installed X.509 certificate), and a locally authenticated user account. The order in which ACL entries are evaluated and applied are based on their position in this list. Use the Move Up and Move Down buttons to change an entry's priority.
Add Button	Displays a dialog box that allows an entry to be added to the Access Control List. The fields in this dialog are populated with default values. 

Certificate ID
When a UCTL Manager specifies a certificate ID from its command line, its corresponding UCTL Server compares it to the certificate ID entered here. If a match is found and the certificate ID has a corresponding entry in the Certificate Map list, the authority to execute the request is granted or denied to the specified local user account.

Local User Account
A local user account that may be provided to UCTL Server for authentication.

Request Type
Specifies the type of request issued by the UCTL Manager. Valid request types are included in the list. * may also be specified to indicate the rule applies to all valid UCTL Manager request types.

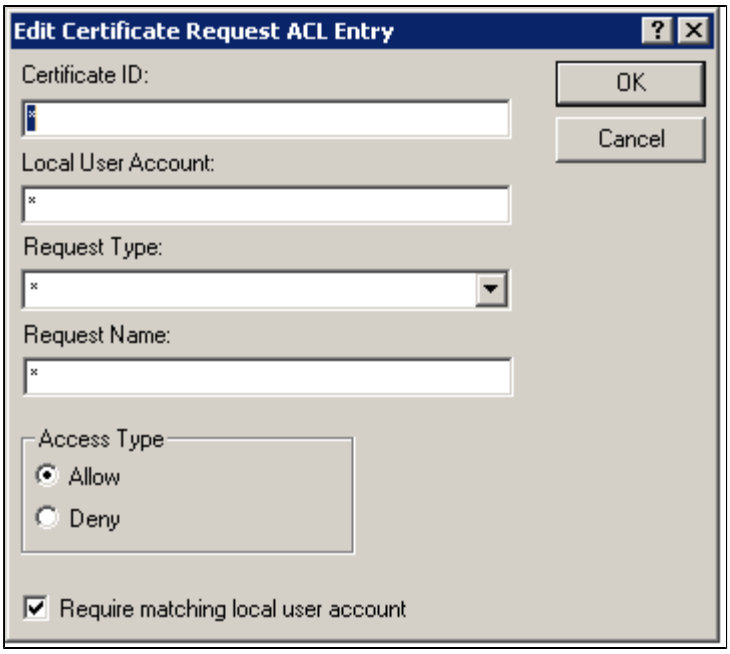
Request Name:
Enter the name of a Universal component here to set access for the specified local user account based on the UCTL Manager request. Use the default value of * to grant or deny access to all requests issued by the user.

Access Type
Will cause the Universal Control Server to accept/Reject incoming connections from the specified host, provided the Universal Control Manager is executed by the specified user account and the process is owned by the specified local user account.

Requires Matching local user account
When this option is checked, a local user account must be supplied to the Universal Control Server that matches the user account in whose security context the Universal Server component is running. This option only applies to Stop requests and is effectively disabled for Refresh and Start requests. To prevent unauthorized users from issuing those requests, make sure security is enabled in the UCtl Server's configuration.

Edit Button

Displays a dialog that allows the selected Access Control List entry to be modified.



Certificate ID
When a UCTL Manager specifies a certificate ID from its command line, its corresponding UCTL Server compares it to the certificate ID entered here. If a match is found and the certificate ID has a corresponding entry in the Certificate Map list, the authority to execute the request is granted or denied to the specified local user account.

Local User Account
A local user account that may be provided to UCTL Server for authentication.

Request Type
Specifies the type of request issued by the UCTL Manager. Valid request types are included in the list. * may also be specified to indicate the rule applies to all valid UCTL Manager request types.

Request Name:
Enter the name of a Universal component here to set access for the specified local user account based on the UCTL Manager request. Use the default value of * to grant or deny access to all requests issued by the user.

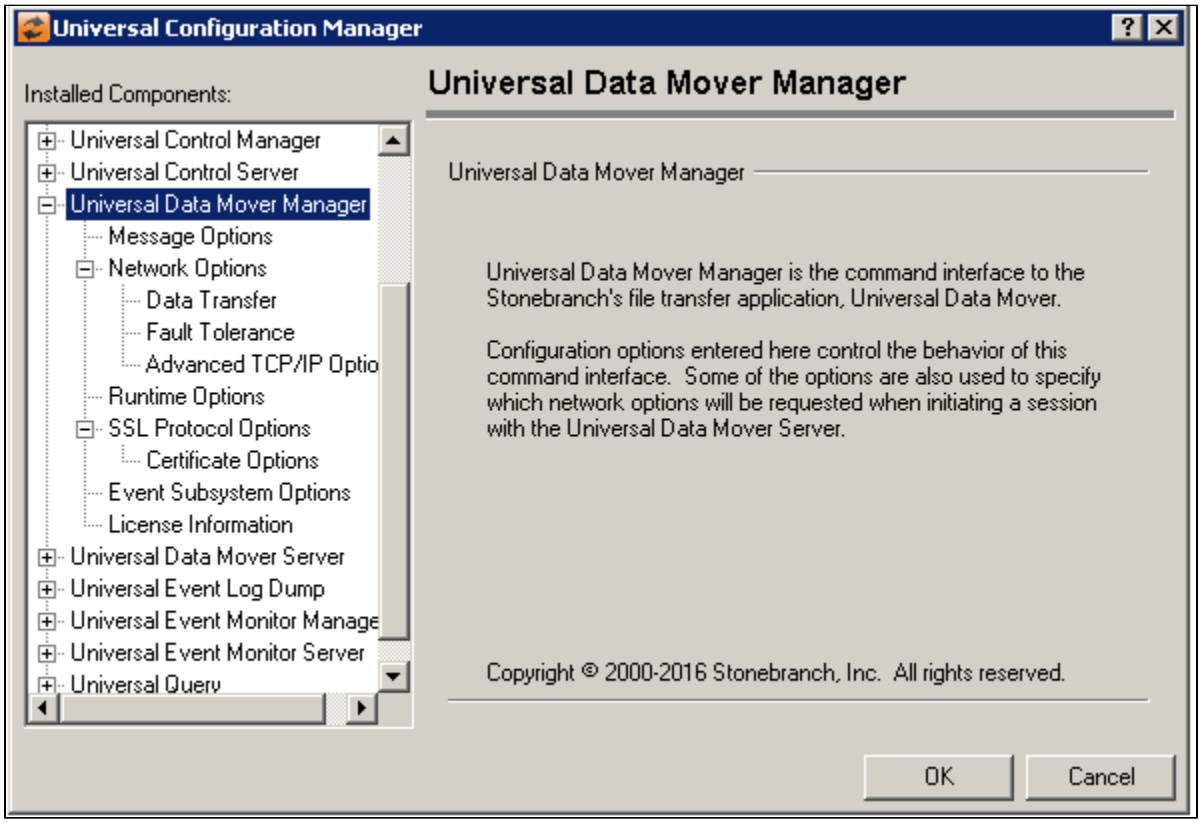
Access Type

	Will cause the Universal Control Server to accept/Reject incoming connections from the specified host, provided the Universal Control Manager is executed by the specified user account and the process is owned by the specified local user account. **Requires Matching local user account** When this option is checked, a local user account must be supplied to the Universal Control Server that matches the user account in whose security context the Universal Server component is running. This option only applies to Stop requests and is effectively disabled for Refresh and Start requests. To prevent unauthorized users from issuing those requests, make sure security is enabled in the UCtl Server's configuration.
Delete Button	Removes the selected item from the Access Control List.
Move Up Button	Moves the selected item up one position in the list.
Move Down Button	Moves the selected item down one position in the list.

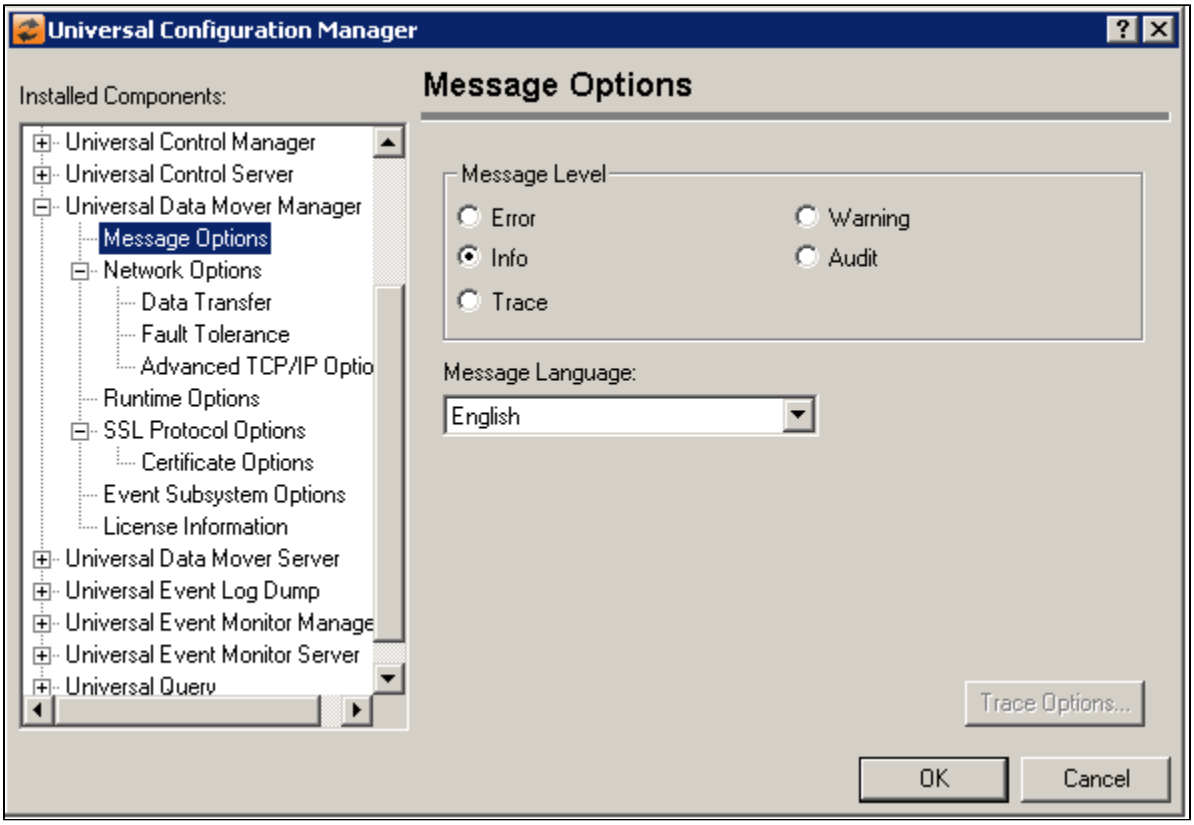
Universal Configuration Manager - Universal Data Mover Manager

- Universal Data Mover Manager
- Universal Data Mover Manager: Message Options
- Universal Data Mover Manager: Network Options
  - Universal Data Mover Manager: Network Options - Data Transfer
  - Universal Data Mover Manager: Network Options - Fault Tolerance
- Universal Data Mover Manager: Network Options - Advanced TCP/IP Options
- Universal Data Mover Manager: Runtime Options
- Universal Data Mover Manager: SSL/TLS Protocol Options
  - Universal Data Mover Manager: SSL/TLS Protocol Options - Certificate Options
- Universal Data Mover Manager: Event Subsystem Options
- Universal Data Mover Manager: License Information

Universal Data Mover Manager

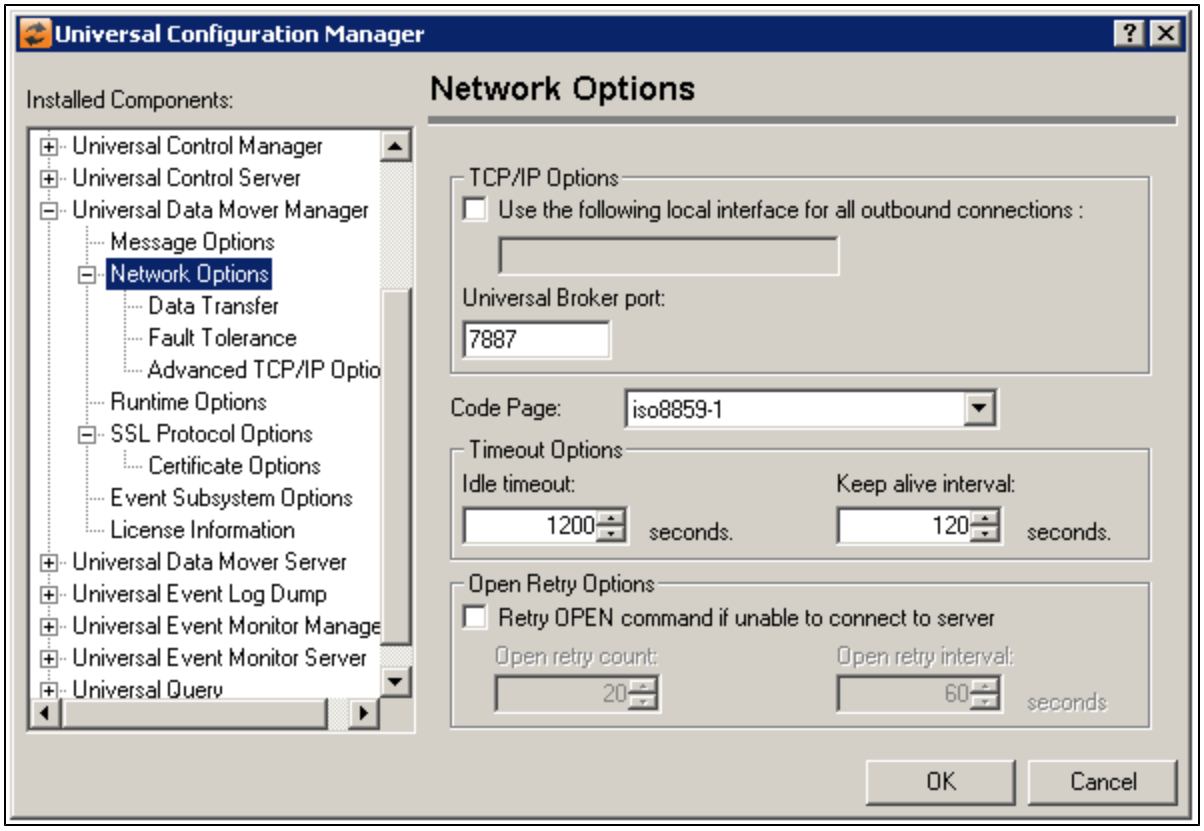


Universal Data Mover Manager: Message Options

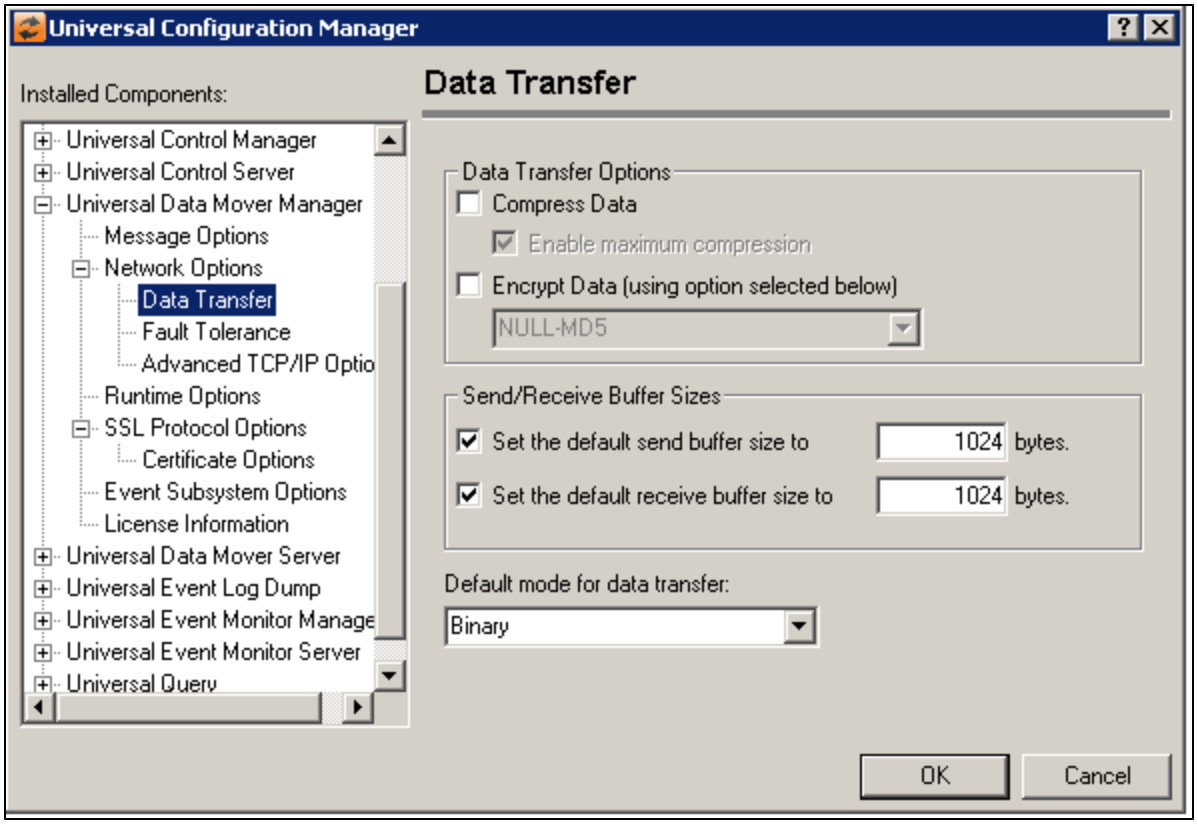


Field	Description
Message Level	**Error** Turns on message logging for errors only. **Warning** Turns on message logging for warnings and errors. **Info** Turns on message logging for all informational, warning and error messages. **Audit** Turns on message logging for all audit, informational, warning and error messages. **Trace** Allows all messages to be logged, including those used for debugging purposes.
Message Language	Specifies the language used to display error messages.

Universal Data Mover Manager: Network Options



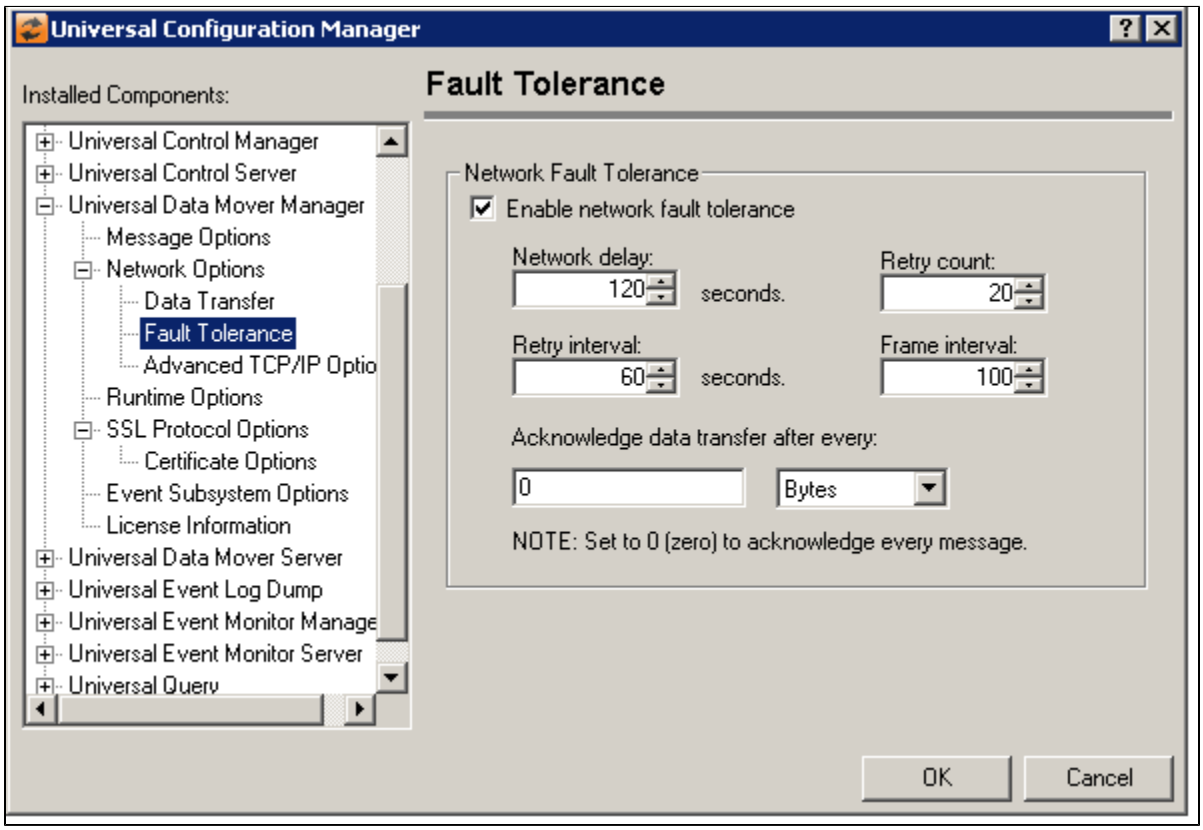
Field	Description
TCP/IP Options	**Use the following local interface for all outbound connections** When this option is selected, the IP address or alias of a local interface may be specified that will be used for all outbound TCP/IP connections established by UDM Manager. If this option is not selected, any available local interface will be used. **Universal Broker Port** Enter a port number or service name that will be used to connect to Universal Broker.
Code Page	Sets the translation table used to translate text-based data transmitted across the network.
Timeout Options	**Idle Timeout** The number of seconds that must elapse during an idle period before timing out. **Keep Alive interval** This value specifies the frequency with which a keep alive message will be sent to the server(s). UDM uses this message to verify that a network connection still exists with the server during periods of network inactivity.
Open Retry Options	**Retry open command if unable to connect to server** Select this option to have UDM retry failed attempts to open a session with a UDM Server. The Retry Count and the Retry Interval values control the number of retry attempts and the interval between these attempts, respectively. **Open retry count** This option specifies the number of times UDM will attempt to open a session with a UDM Server. This value is only used when the open retry option is enabled. **Open retry interval** This option specifies the amount of time, in seconds, a UDM Manager will wait between attempts to open a session with a UDM Server. This value is only used when the open retry option is enabled.



Field	Description
Data Transfer Options	**Compress Data** Controls compression of data transmitted across the network. Select the checkbox to turn this option on. **Encrypt Data (Using option selected below)** Controls encryption of data transmitted across the network. Select the checkbox to turn this option on. This option sets a default value that is used when the encrypt parameter of the open statement is omitted. The configured default may be overridden at any time from the open statement. When this value is set to a specific SSL/TLS cipher, UDM always requests that cipher when encrypting data. Note that if the target UDM Server does not support the selected cipher (as indicated by the Server's configured data SSL/TLS cipher list), the encryption request fails. When this option is checked and no specific cipher is specified, UDM uses the Manager and Server's configured data SSL/TLS cipher list to negotiate the encryption method. When the data encryption checkbox is selected, you may select an item in this list to set the UDM Manager's encrypt configuration option to an SSL/TLS cipher. UDM then requests this cipher whenever it encrypts the data it transfers. Note that if the target UDM Server does not support the selected cipher (as indicated by the Server's configured data SSL/TLS cipher list), the encryption request fails. The following cipher suites are available: • RC4-SHA -- 128-bit RC4 encryption, SHA-1 message digest • RC4-MD5 -- 128-bit RC4 encryption, MD5 message digest • AES256-SHA -- 256-bit AES encryption, SHA-1 message digest • AES128-SHA -- 128-bit AES encryption, SHA-1 message digest • DES-CBC3-SHA -- 128-bit Triple-DES encryption, SHA-1 message digest • DES-CBC-SHA -- 128-bit DES encryption, SHA-1 message digest • NULL-SHA -- No encryption, SHA-1 message digest • NULL-MD5 -- No encryption, MD5 message digest • NULL-NULL -- Disables the SSL protocol When the data encryption option is checked and the (negotiate using data SSL/TLS cipher list) item is selected, the encrypt option is set to yes, which instructs UDM Manager to negotiate the encryption method with the Server, using their respective data SSL/TLS cipher lists.
Send/Receiv	**Set Default Send Buffer size to** When this option is selected, a value may be specified that overrides the default size of the application buffer used to send data

e Buffer Sizes	across a network connection. A buffer size of 0 (zero) is saved when this option is left un-selected. UDM will then manage the size of the buffers itself based on internal limits. Exercise caution when changing this option, as it could have a significant performance impact. See the send_buffer_size configuration option in the UDM Reference Manual for more information. **Set default receive buffer size to** When this option is selected, a value may be specified that overrides the default size of the application buffer used to retreive data from a network connection. A buffer size of 0 (zero) is saved when this option is left un-selected. UDM will then manage the size of the buffers itself based on internal limits. Exercise caution when changing this option, as it could have a significant performance impact. See the recv_buffer_size configuration option in the UDM Reference Manual for more information.
Default mode for data transfer	Specifies the default manner in which files will be transferred (Binary or Text).

Universal Data Mover Manager: Network Options - Fault Tolerance



Field	Description
Network Fault Tolerance	**Enable Network Fault Tolerance** When this option is selected, network fault tolerance will be requested for each instance of a Universal Data Mover Manager that is run from this location. This means that if the remote Universal Data Mover Server supports network fault tolerance, all lost network connections will be detected and recovered. **Network Delay** The number of seconds that can be attributed to network latency for any transmission of data among Universal components. This value is automatically added to any network operation where a time out condition is possible and needs to be handled. **Retry Count** The maximum number of reconnects that will be attempted if a network connection is lost and network fault tolerance is enabled. **Retry Interval** The number of seconds between each reconnect attempt.

Frame interval
The frame interval is the number of transmission blocks between synchronization points.

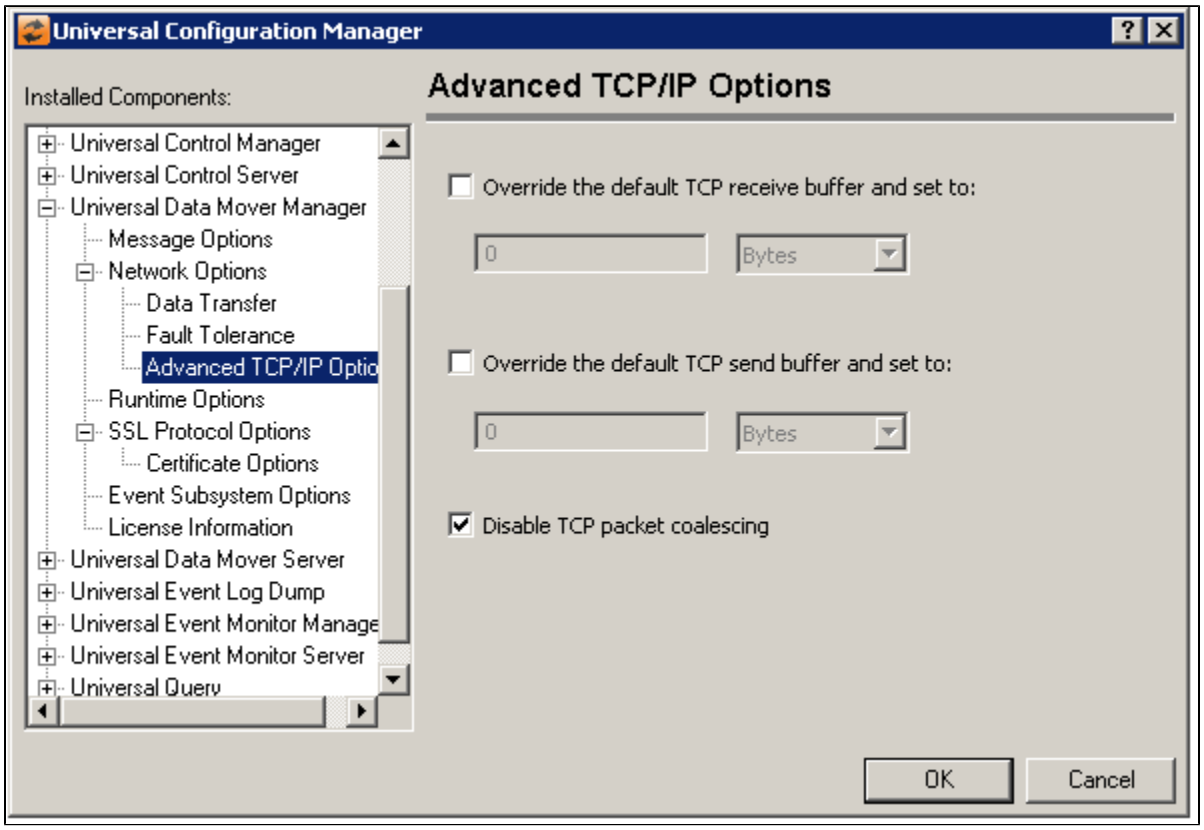
Acknowledge data transfer after every
Specifies the size of the Network Fault Tolerant (NFT) acknowledgement window. The NFT protocol requires an acknowledgement (ACK) message from the receiver to acknowledge the receipt of the data. The ACK window specifies the amount of data sent before an ACK is required. The larger the window the more data that is sent before waiting for an ACK message. A large ACK window can improve transfer performance over large bandwidth, high latency connections.

The format of the value is nnnnb, where 'nnnn' is a numeric value and 'b' is one of the following optional unit specifiers:

- b - bytes (the default, used if no unit is specified)
- k - kilobytes, where 1 kb = 1024 bytes
- m - megabytes, where 1 mb = 1048576 bytes

The maximum value permitted is 500M. A value of 0 (zero) causes every message that contains data for a file transfer to be acknowledged (default NFT behavior).

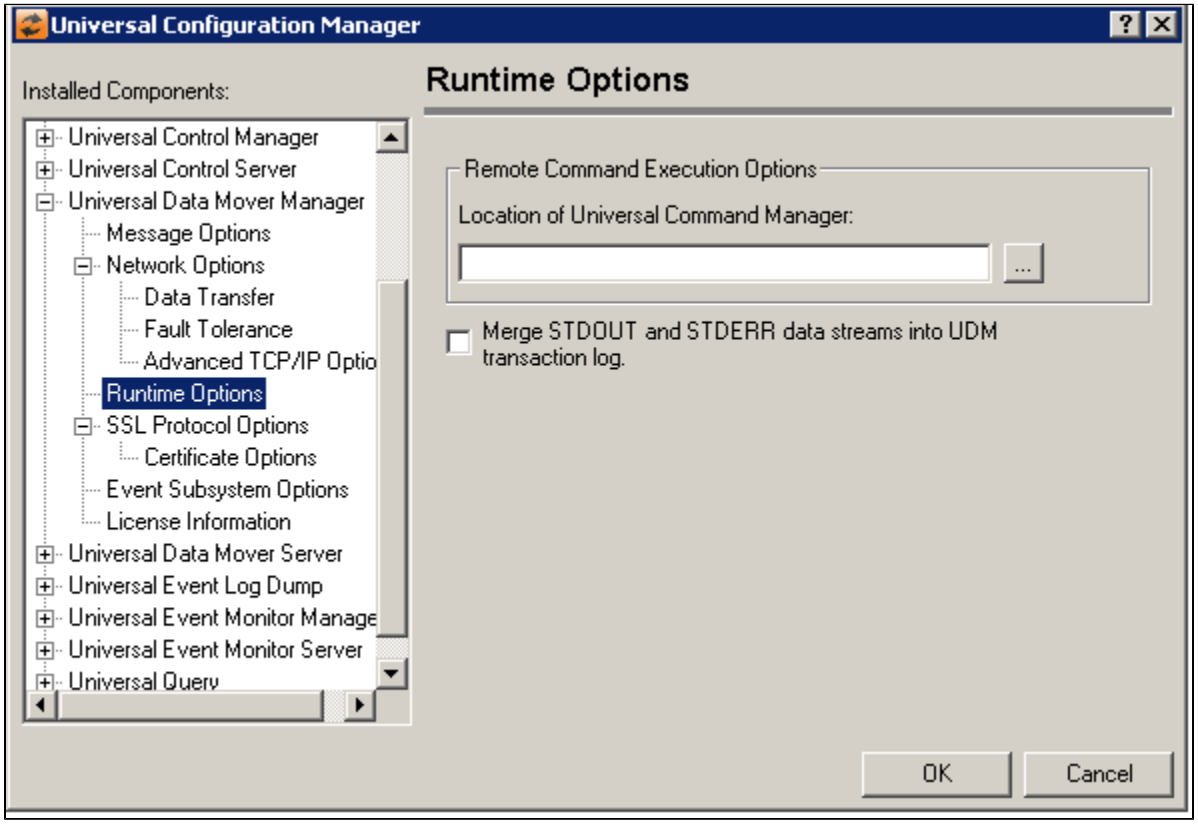
Universal Data Mover Manager: Network Options - Advanced TCP/IP Options



Field	Description
Override the default TCP receive buffer and set to	Allows the UDM manager to set the TCP receive buffer for a UDM server larger than the default maximum of 64k bytes. Setting this value enables RFC 1323 window scaling support, which can increase TCP buffer sizes over a fast, high-bandwidth network to almost 1 gigabyte (1,073,725,400 bytes). When this option is not selected a UDM uses a default of 0 bytes, which disables RFC 1323 window scaling support. Note that even when a value is specified for this option, the TCP/IP stack ultimately decides the size of the transfer buffer. Specifies the TCP receive buffer size for the TCP socket connections from the UDM manager to the UDM server. The option has no effect for UDM three party transfers. The UDM server and Universal Broker configuration affects UDM three party transfers. Refer to the Universal Broker tcp_recv_buffer option to set the TCP receive buffer for the other end of the socket connection. The socket TCP receive buffer is used by TCP to determine the size of the TCP window it advertises to the sending TCP. If both TCP implemenations support window scaling (as defined in RFC 1323), buffer sizes greater then 65535 will be used for the TCP window size. Note that both the sending and receiving TCP implemenations must support window scaling for it to be used in the TCP connection.

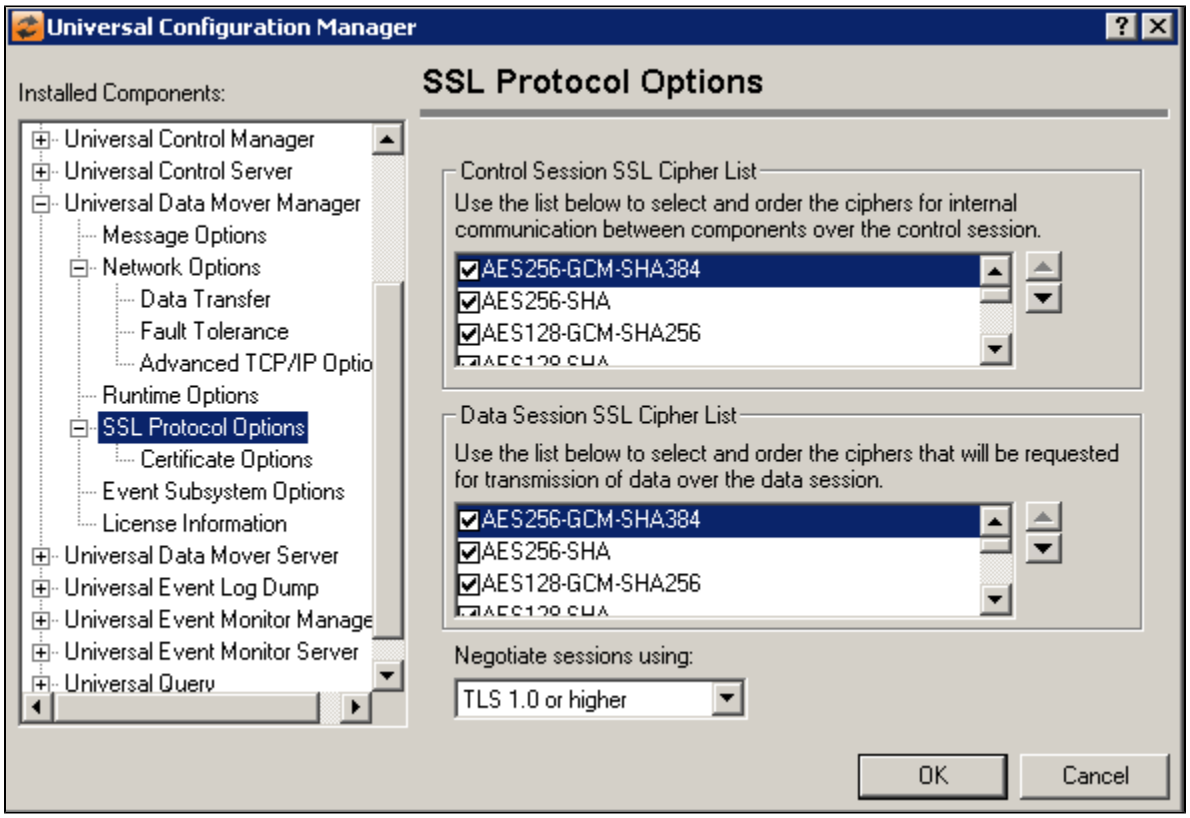
	Adjusting the TCP receive buffer size may improve bulk data transfer performance for certain types of transmission links when transfering data from the UDM server to the UDM manager. The format of the value is nnnnb, where 'nnnn' is a numeric value and 'b' is one of the following optional unit specifiers: • b - bytes (the default, used if no unit is specified) • k - kilobytes, where 1 kb = 1024 bytes • m - megabytes, where 1 mb = 1048576 bytes • g - gigabytes, where 1 gb = 1073741824 bytes Note that the maximum allowable value for this option is 1073725440 bytes, which can be requested using a value of 1g.
Override the default TCP send buffer and set to	Specifies the TCP send buffer size for the TCP socket connections from the UDM manager to the UDM server. The option has no effect for UDM three party transfers. The UDM server and Universal Broker configuration affects UDM three party transfers. Refer to the Universal Broker tcp_send_buffer option to set the TCP send buffer for the other end of the socket connection. Adjusting the TCP send buffer size may improve bulk data transfer performance for certain types of transmission links when transfering data from the UDM manager to UDM server. The format of the value is nnnnb, where 'nnnn' is a numeric value and 'b' is one of the following optional unit specifiers: • b - bytes (the default, used if no unit is specified) • k - kilobytes, where 1 kb = 1024 bytes • m - megabytes, where 1 mb = 1048576 bytes • g - gigabytes, where 1 gb = 1073741824 bytes Note that the maximum allowable value for this option is 1073725440 bytes, which can be requested using a value of 1g.
Disable TCP packet coalescing	Select this option to disable the TCP packet coalescing algorithm (that is, Nagle algorithm). When this option is unchecked, the default TCP implementation is used, which may or may not disable the algorithm. Exercise caution when changing this option, as it could have a significant performance impact. See the tcp_no_delay configuration option in the UDM Reference Manual for more information. This option is checked by default.

Universal Data Mover Manager: Runtime Options



Field	Description
Remote Command Execution Options	**Location of Universal Command Manager** If you have a licensed version of Universal Command Manager v311 or later installed on this system, you may execute system commands on remote machines using the UDM Manager exec command. Enter the location of the Universal Command Manager application file (ucmd.exe) if it resides outside of your system path. Otherwise, this value may be left blank.
Merge STDOUT and STDERR data streams into UDM transaction log	Select this option to cause all information written to the STDOUT and STDERR data streams to be merged into the UDM transaction log.

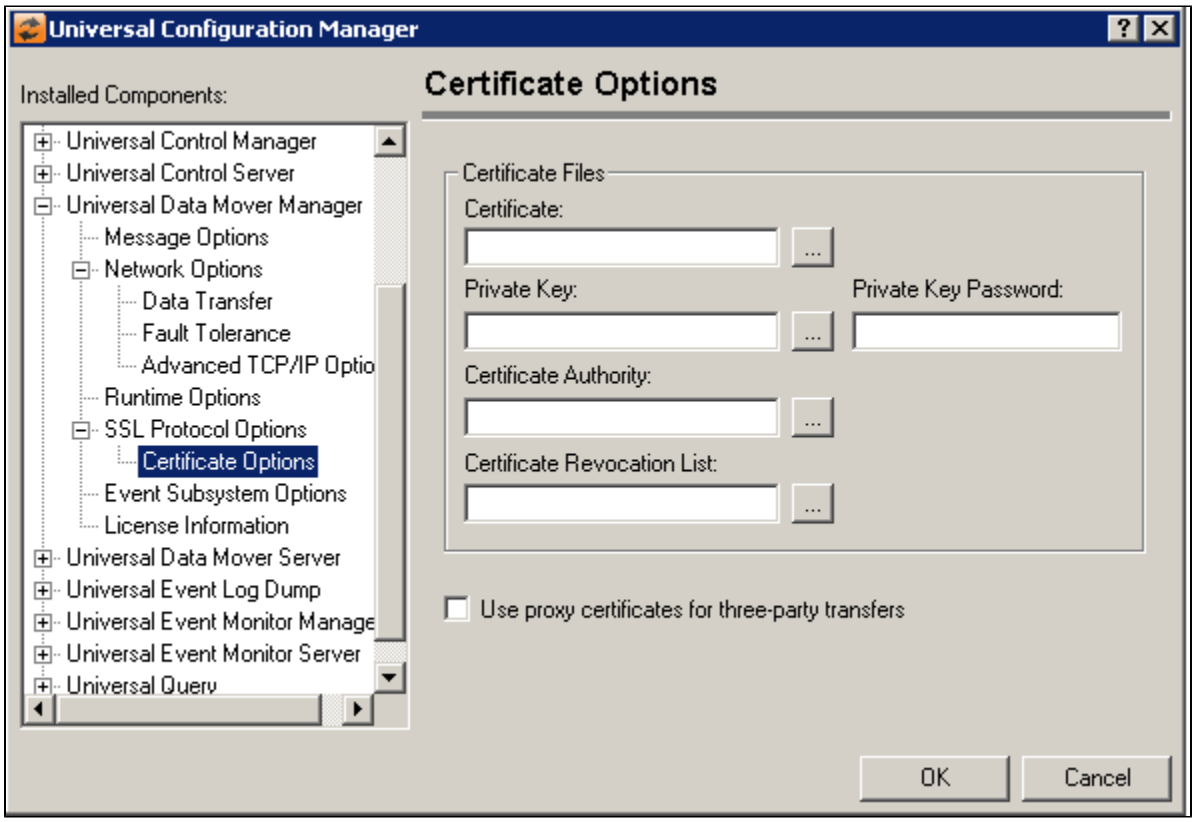
Universal Data Mover Manager: SSL/TLS Protocol Options



Field	Description
Control Session SSL cipher List	Universal Data Mover uses the SSL/TLS protocol for internal communication between application components. A control session is established within the application that provides the connections necessary for this communication to take place. The SSL/TLS cipher suites available for communication across the control session are listed here. The selected cipher suites will be forwarded to the Universal Data Mover Server. The Server will compare this list to its own list of ciphers it is capable of accepting. The result of the comparison will be an agreed-upon cipher used for communication over the control session. If you are unsure which ciphers will be accepted by the Server, it is best to leave them all selected. Use the up/down arrows on the right to sort the list and establish a priority for the selected ciphers. The first one in the list that is accepted by the Server will be the one used. The following cipher suites are available: • RC4-SHA -- 128-bit RC4 encryption, SHA-1 message digest • RC4-MD5 -- 128-bit RC4 encryption, MD5 message digest • AES256-SHA -- 256-bit AES encryption, SHA-1 message digest • AES128-SHA -- 128-bit AES encryption, SHA-1 message digest • DES-CBC3-SHA -- 128-bit Triple-DES encryption, SHA-1 message digest • DES-CBC-SHA -- 128-bit DES encryption, SHA-1 message digest • NULL-SHA -- No encryption, SHA-1 message digest • NULL-MD5 -- No encryption, MD5 message digest • NULL-NULL -- Disables the SSL protocol

	To disable SSL/TLS encryption and message authentication for messages transmitted across the control session, select the NULL-NULL entry. Use of this cipher requires that the remote UDM Server's encrypt_control_session option be set to no. Otherwise, the UDM Server will issue a protocol error when the Manager tries to establish the control session with it.
Data Session SSL Cipher List	Universal Data Mover uses the SSL/TLS protocol for transmission of data across the network. A data session is established within the application that provides the connections necessary for the transfer of data to take place. The SSL/TLS cipher suites available for communication across the data session are listed here. The selected cipher suites will be forwarded to the Universal Data Mover Server. The Server will compare this list to its own list of ciphers it is capable of accepting. The result of the comparison will be an agreed-upon cipher used for communication over the data session. If you are unsure which ciphers will be accepted by the Server, it is best to leave them all selected. Use the up/down arrows on the right to sort the list and establish a priority for the selected ciphers. The first one in the list that is accepted by the Server will be the one used. • RC4-SHA -- 128-bit RC4 encryption, SHA-1 message digest • RC4-MD5 -- 128-bit RC4 encryption, MD5 message digest • AES256-SHA -- 256-bit AES encryption, SHA-1 message digest • AES128-SHA -- 128-bit AES encryption, SHA-1 message digest • DES-CBC3-SHA -- 128-bit Triple-DES encryption, SHA-1 message digest • DES-CBC-SHA -- 128-bit DES encryption, SHA-1 message digest • NULL-SHA -- No encryption, SHA-1 message digest • NULL-MD5 -- No encryption, MD5 message digest • NULL-NULL -- Disables the SSL/TLS protocol
Negotiate sessions using	(No help provided.)

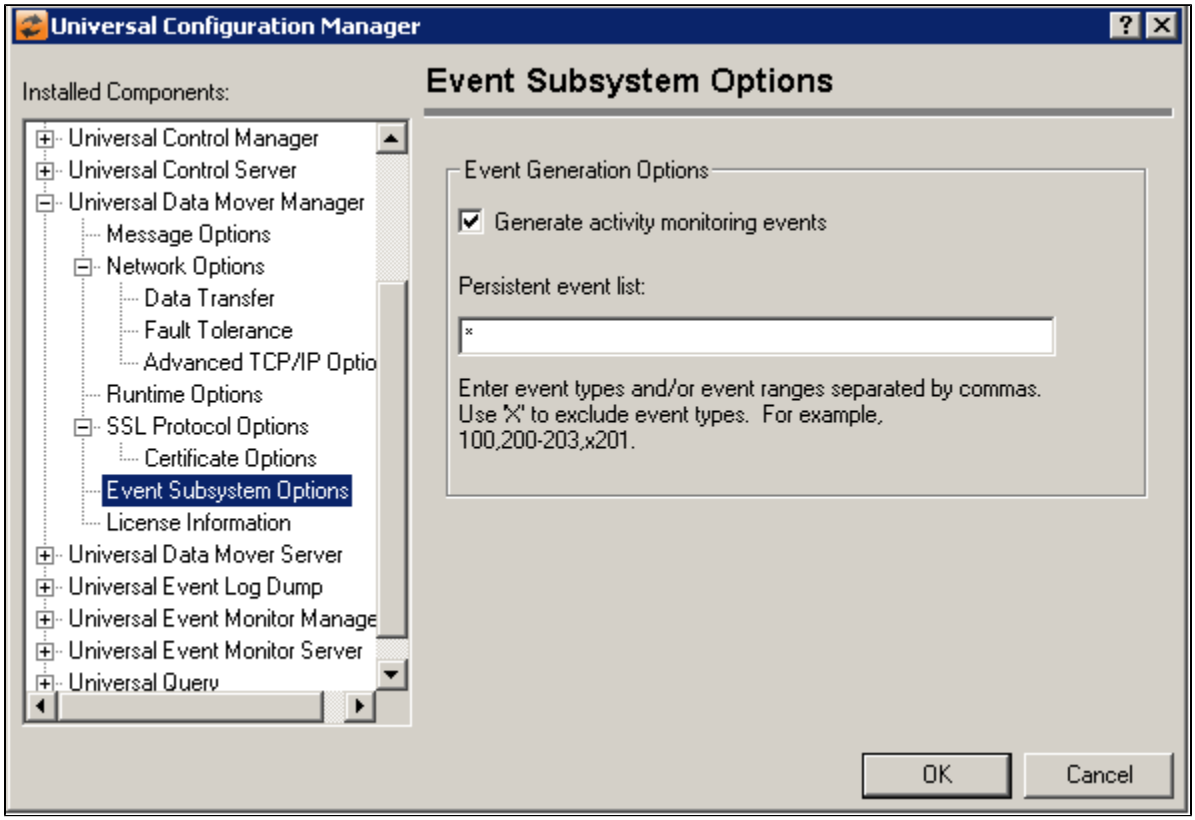
Universal Data Mover Manager: SSL/TLS Protocol Options - Certificate Options



Field	Description
Certificate Files	**Certificate** The Certificate option specifies the X.509 certificate file that identifies the program to remote components with which it connects using the SSL/TLS protocol.

	Private Key The Private key option specifies the private key file associated with the X.509 certificate specified in the Certificate file. **Private Key Password** The Private key password specifies the password associated with the private key file. **Certificate Authority** The Certificate Autority option specifies a file with a list of trusted authorities. Any certificate received that is signed by a trusted Certificate Authority is accepted. **Certificate Revocation List** The Certificate Revocation List option specifies a file with a list of revoked certificates. Any certificate received that is in this list will not be accepted.
Use proxy certificates for third-party transfers	Select this option to use proxy certificates to establish a three-party transfer session. This option is unchecked by default.

Universal Data Mover Manager: Event Subsystem Options

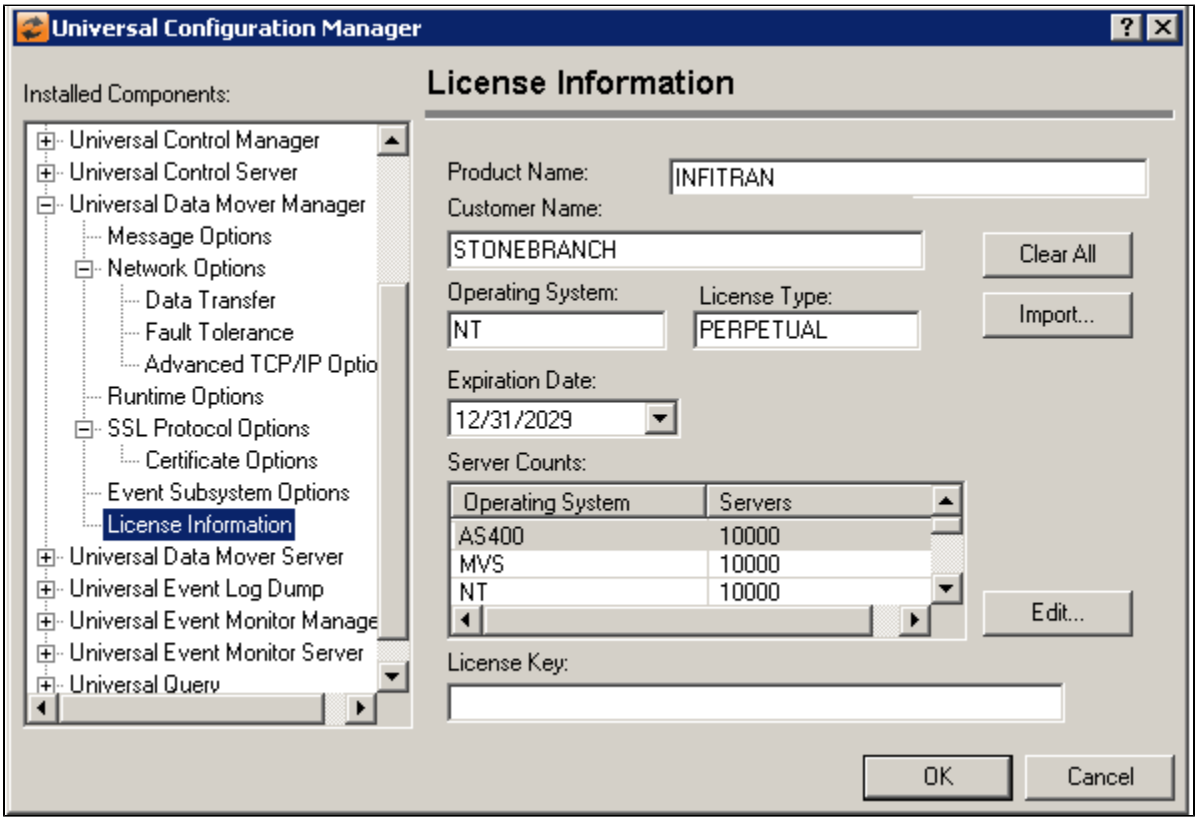


Field	Description
Event Generation Options	**Generate activity monitoring events** When this option is selected, the Universal Data Mover Manager will generate events that monitor product activity. This option is selected by default. **Persistent event list** Specifies which events are to be generated and processed as persistent events. A persistent event is saved in a Universal Enterprise Controller (UEC) event database for long-term storage. Enter a list and/or range of numeric event types, separated by a comma. To exclude an event or range of events, precede the entry with an upper- or lower-case 'X'. Use an asterisk ***** to specify all event types. For example: • x* Do not generate any events (this is the default)

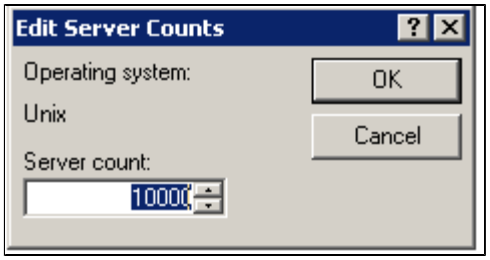
- *
  Generate all event types
- 100
  Generate all event types except 100
- 100,200-205
  Generate event type 100 and 200 through 205
- 100,200-205,x202
  Generate event types 100 and 200 thru 205, excluding event type 202

Entries are processed from left to right, and the list is order dependent. For example, an entry of x*,100 prevents generation of all event types except 100, while an entry of 100,x* prevents generation of all event types (the right-most entry takes precedence).

Universal Data Mover Manager: License Information



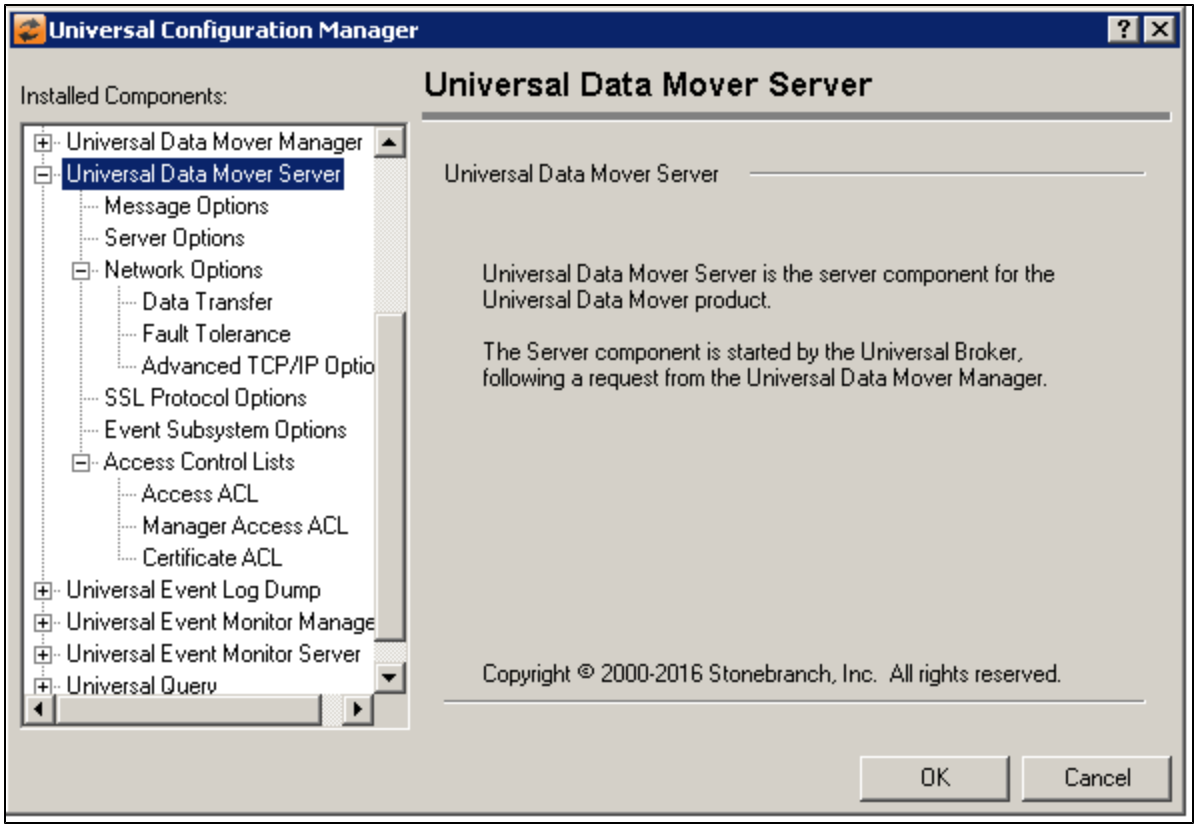
Field	Description
Product Name	Specifies the name of the registered product for which the license key was issued.
Customer Name	Contains the name under which the software is registered. This value must be entered exactly as the customer name specified in the licensing material you received with the software.
Operating System	Identifies the operating system on which your licensed Universal Data Mover Manager will run. This value should be match the operating system specified in the licensing material you received with your software.
License Type	Specifies the type of license issued for the current product. This value is used with the expiration date to determine the length of time the software license is valid.
Expiration Date	Contains the date when the current license for this product will expire. The date entered must match the expiration date specified in the licensing material you received with the software. To change the date, enter a new value, or click the down arrow to display a calendar control.
Server Counts	Contains the number of Universal Data Mover Servers you have licensed, by operating system. This values in this list should match the numbers specified in the licensing material you received with your software.

License Key	Contains the license key that is generated for each product, customer, license type and expiration date. The value contained in this field must match the key specified in the licensing material you received with the software package.
Clear All Button	(No help provided.)
Edit Button	Displays a dialog from which the number of licensed servers for the selected operating system can be updated. 
Import Button	Select this button to browse for and load the Universal Command Manager license file, which will automatically import the license information.

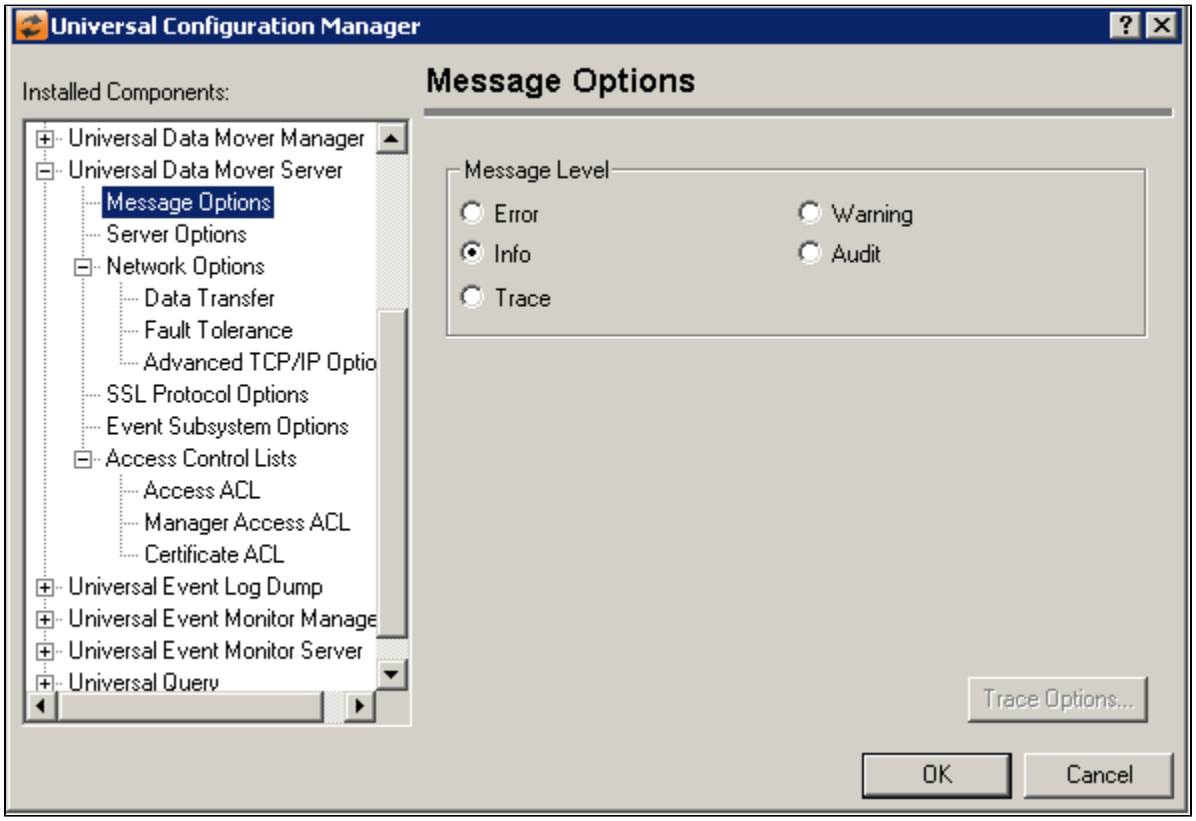
Universal Configuration Manager - Universal Data Mover Server

- Universal Data Mover Server
- Universal Data Mover Server: Message Options
- Universal Data Mover Server: Server Options
- Universal Data Mover Server: Network Options
  - Universal Data Mover Server: Network Options - Data Transfer
  - Universal Data Mover Server: Network Options - Fault Tolerance
  - Universal Data Mover Server: Network Options - Advanced TCP/IP Options
- Universal Data Mover Server: SSL/TLS Protocol Options
- Universal Data Mover Server: Event Subsystem Options
- Universal Data Mover Server: Access Control Lists
  - Universal Data Mover Server: Access Control Lists - Access ACL
  - Universal Data Mover Server: Access Control Lists - Manage Access ACL
  - Universal Data Mover Server: Access Control Lists - Certificate ACL

Universal Data Mover Server



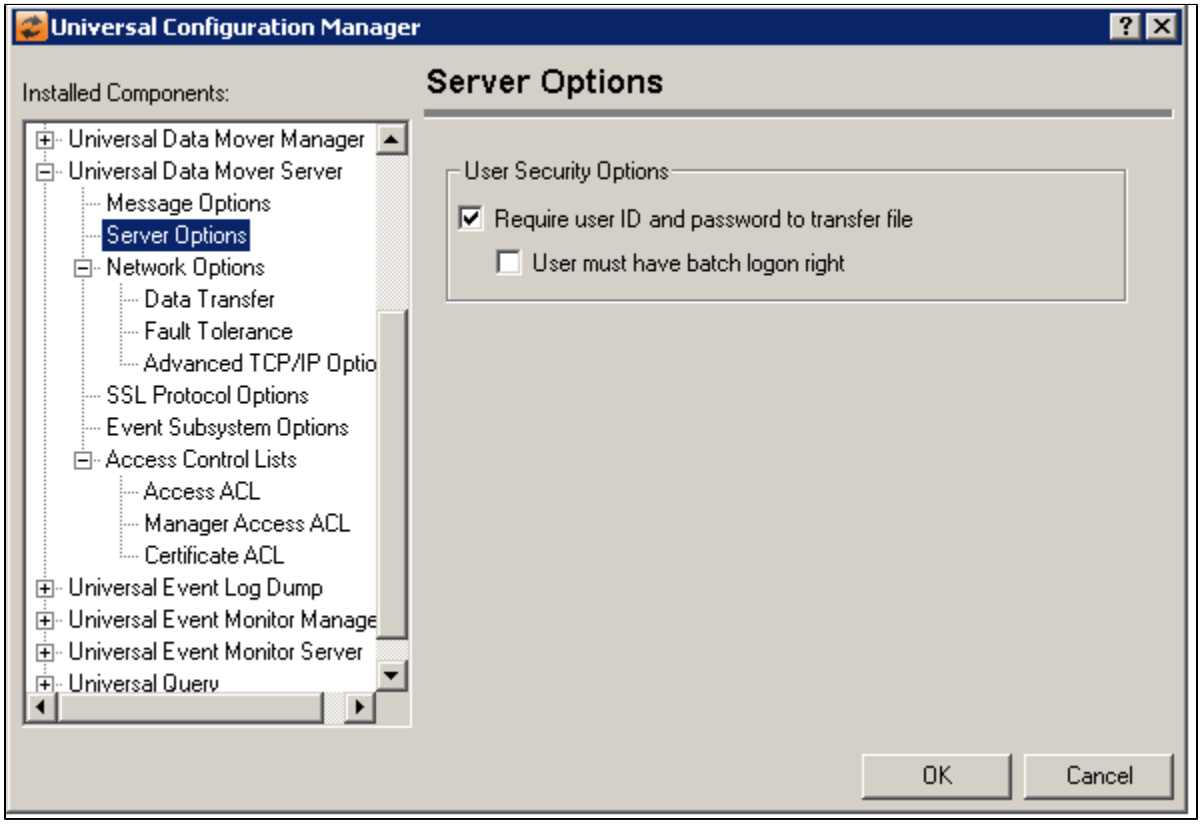
Universal Data Mover Server: Message Options



Field	Description
Message Level	

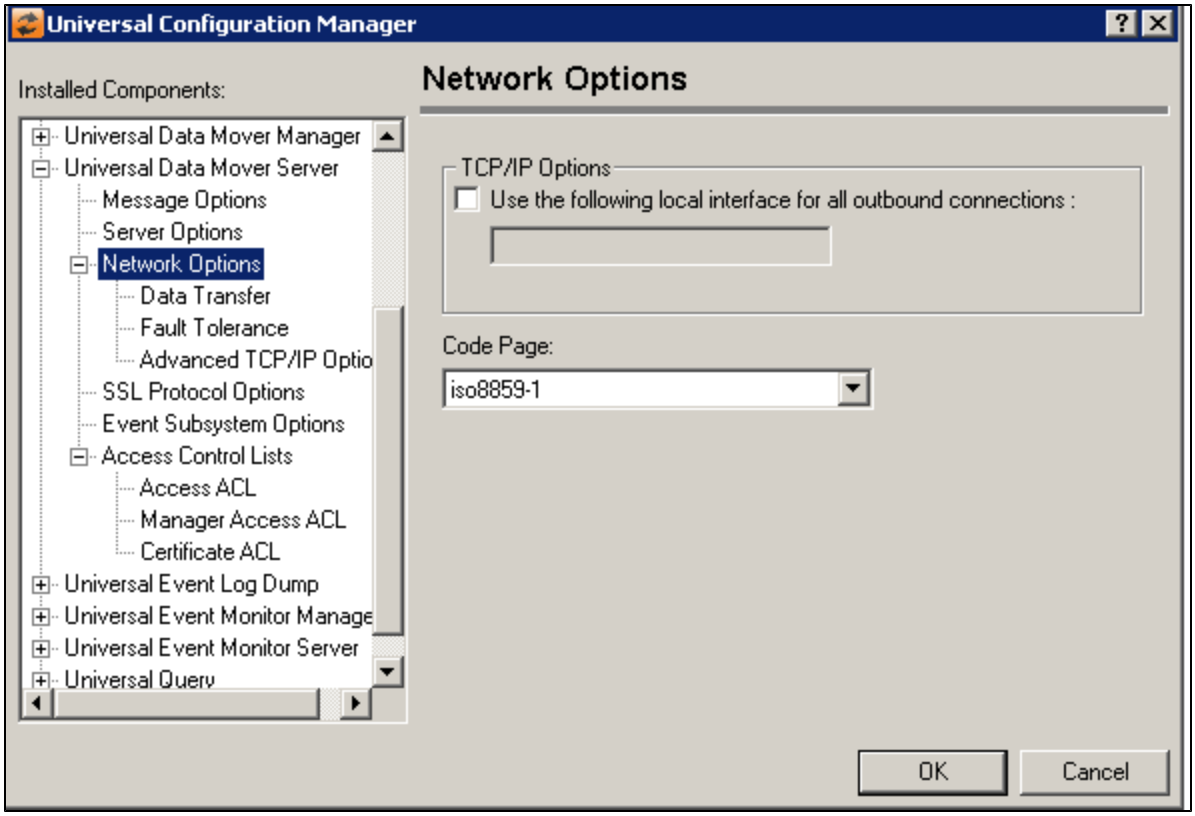
	Error Turns on message logging for errors only. **Warning** Turns on message logging for warnings and errors. **Info** Turns on message logging for all informational, warning and error messages. **Audit** Turns on message logging for all audit, informational, warning and error messages. **Trace** Allows all messages to be logged, including those used for debugging purposes.

Universal Data Mover Server: Server Options

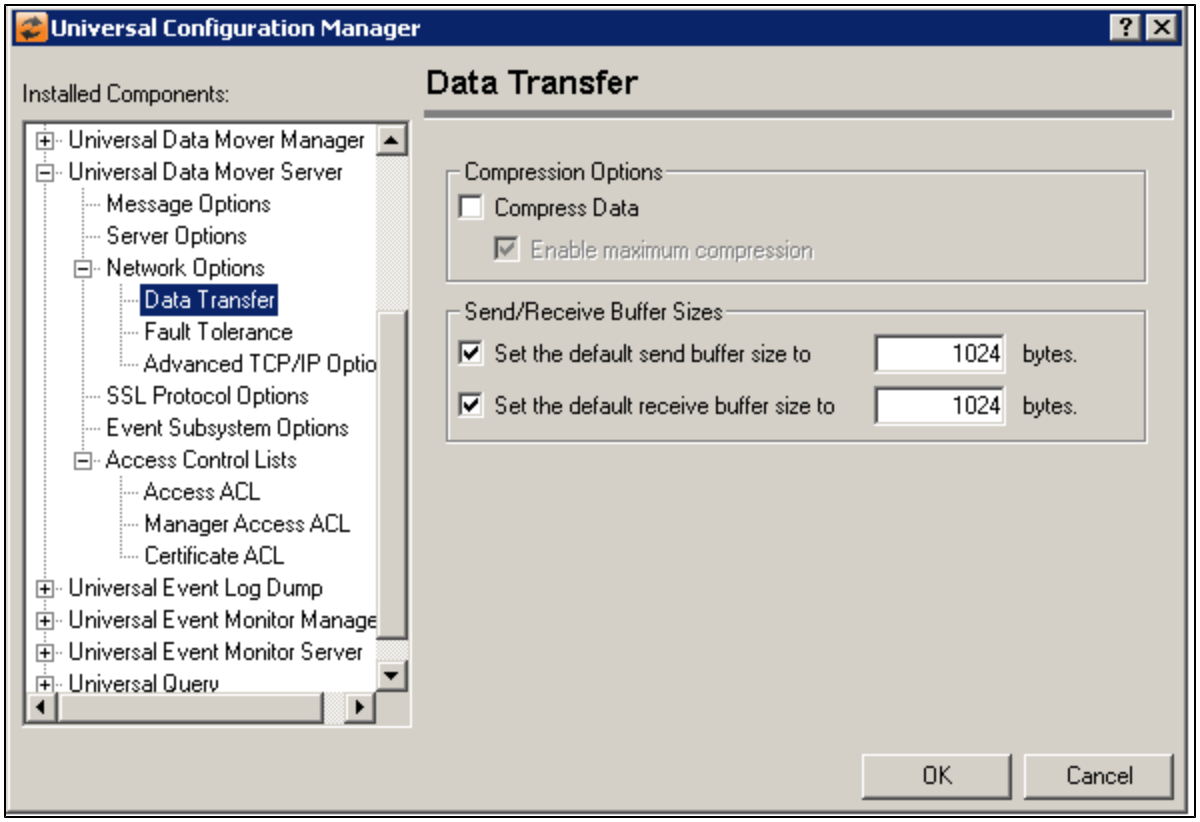


Field	Description
User Security Options	**Require user ID and password to transfer file** When this option is checked, Universal Data Mover will require a valid user ID and password to transfer a file. **User must have batch logon right** When this option is checked, the user account used to open a Universal Data Mover session must be granted the Log on as a batch job privilege.

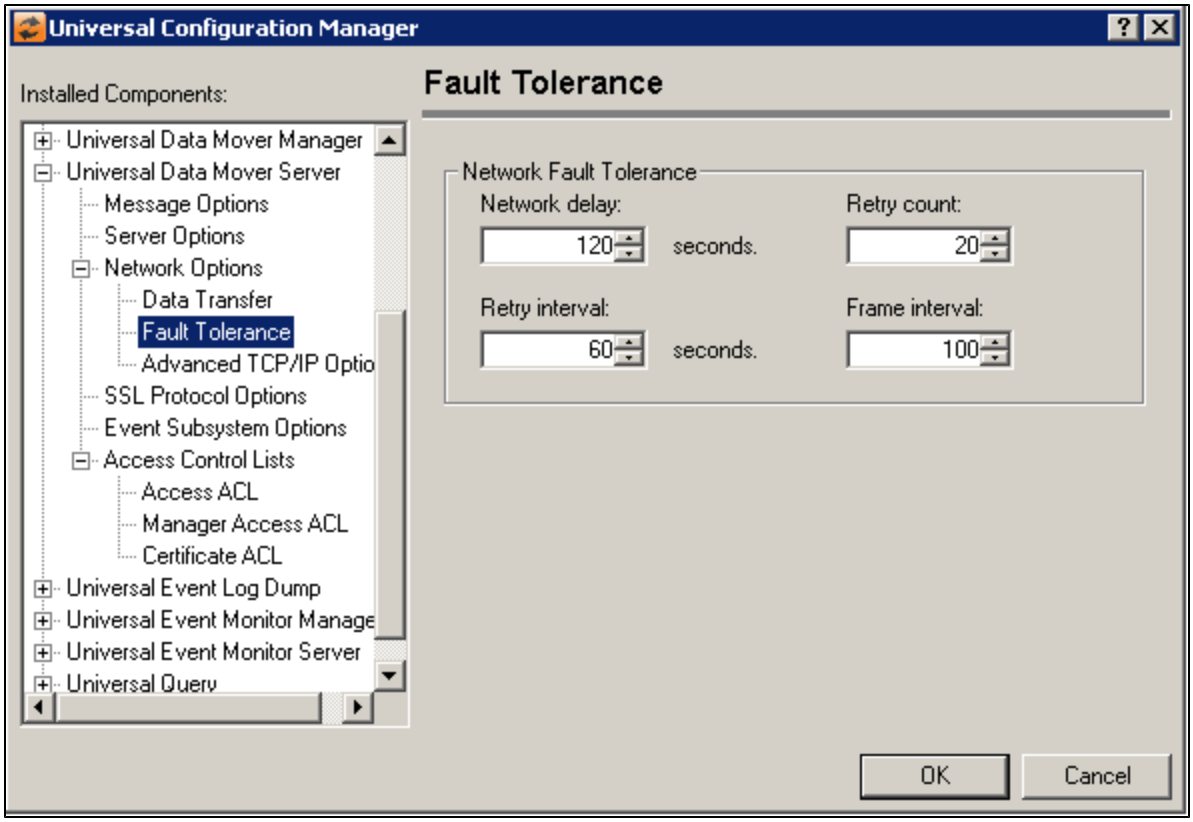
Universal Data Mover Server: Network Options



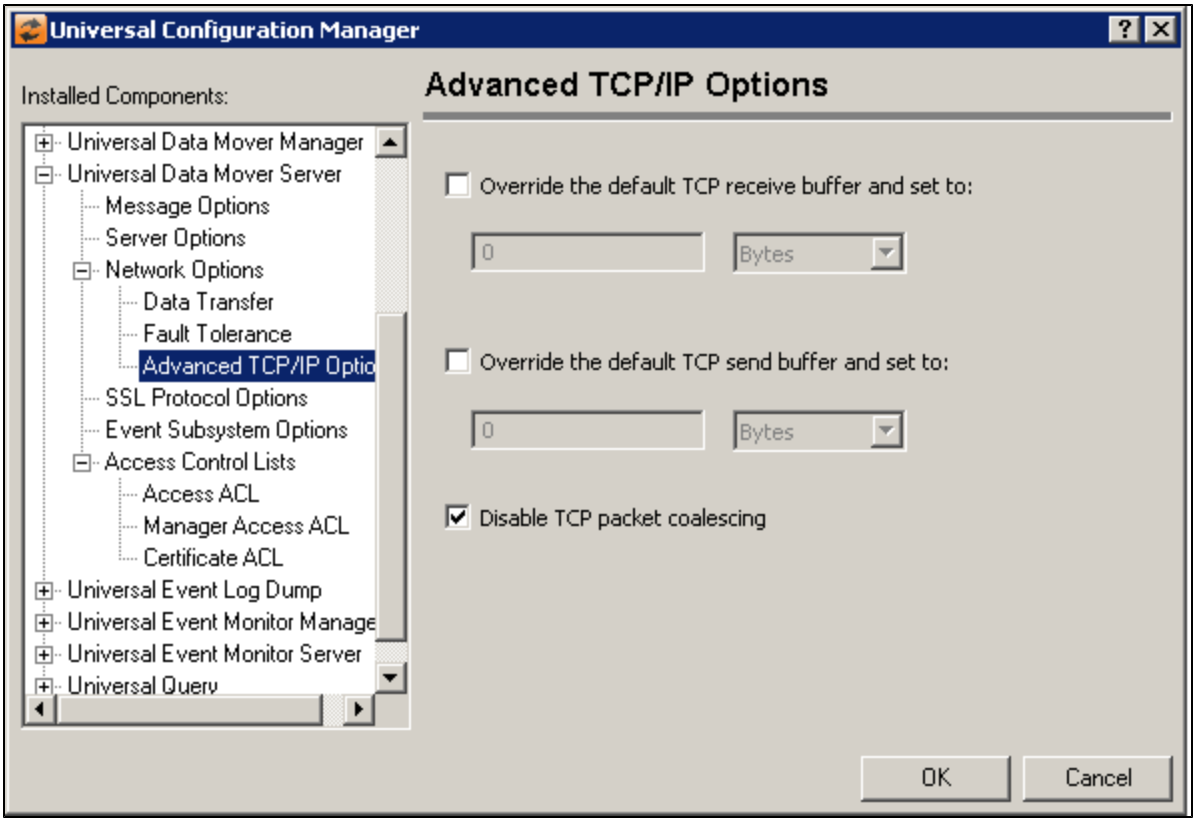
Field	Description
TCP /IP Options	**Use the following local interface for all outbound connections** When this option is selected, the IP address or alias of a local interface may be specified that will be used for all outbound TCP/IP connections established by UDM Server. If this option is not selected, any available local interface will be used. The input box specifies the IP address or alias of a local interface that should be used for all outbound TCP/IP connections established by UDM Server. To have UDM Server use any local interface available, leave this value blank.
Code Page	Sets the translation table used to translate text-based data transmitted across the network.



Field	Description
Compression Options	**Compress Data** When this option is selected, the IP address or alias of a local interface may be specified that will be used for all outbound TCP/IP connections established by UDM Server. If this option is not selected, any available local interface will be used. **Enable maximum compression** Select this option to use the zlib compression algorithm, which usually results in a very high compression rate. When this option is not selected, the hasp compression algorithm will be used. This algorithm is not as CPU-intensive as the zlib algorithm, but the resulting data is a little less compressed.
Send Receive Buffer Sizes	**Set Default send buffer size to** When this option is selected, a value may be specified that overrides the default size of the application buffer used to send data across a network connection. A buffer size of 0 (zero) is saved when this option is left un-selected. UDM will then manage the size of the buffers itself based on internal limits. Exercise caution when changing this option, as it could have a significant performance impact. See the send_buffer_size configuration option in the UDM Reference Manual for more information. **Set Default receive buffer size to** When this option is selected, a value may be specified that overrides the default size of the application buffer used to retreive data from a network connection. A buffer size of 0 (zero) is saved when this option is left un-selected. UDM will then manage the size of the buffers itself based on internal limits. Exercise caution when changing this option, as it could have a significant performance impact. See the recv_buffer_size configuration option in the UDM Reference Manual for more information.



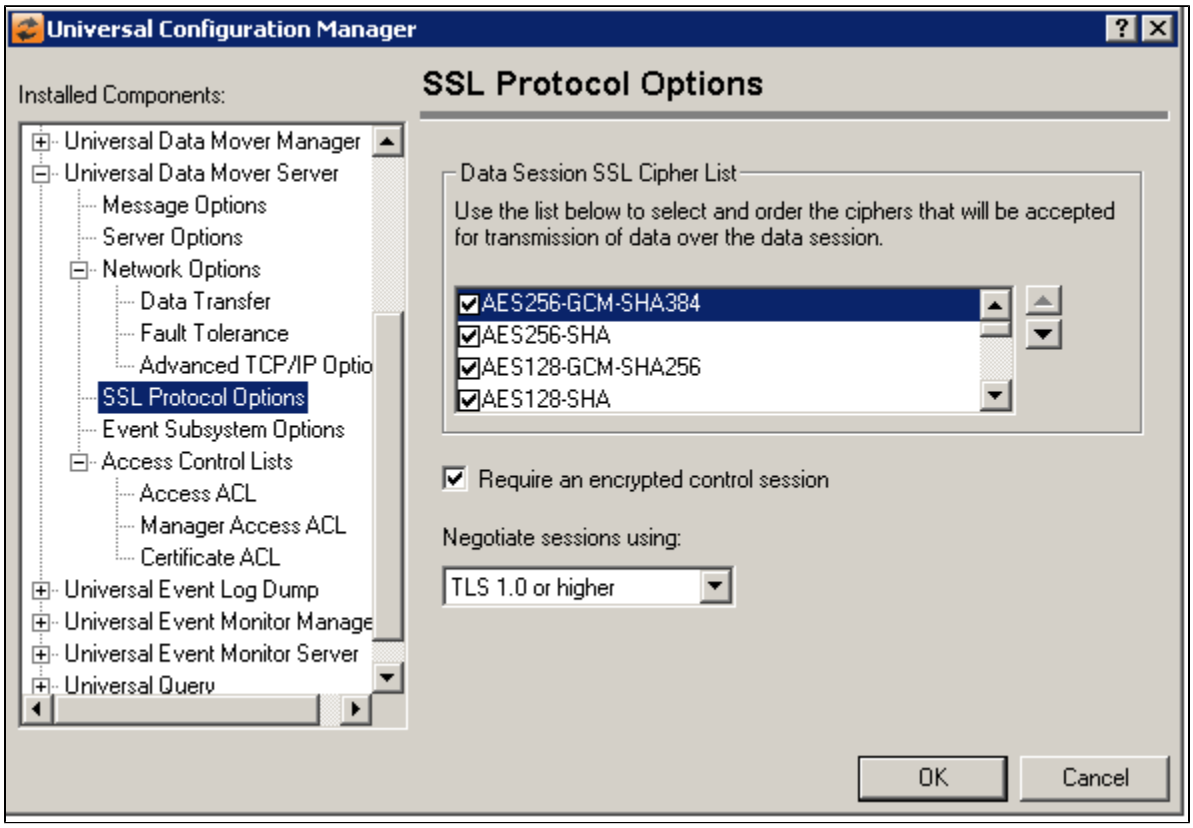
Field	Description
Network Fault Tolerance	**Network delay** The number of seconds that can be attributed to network latency for any transmission of data among Universal components. This value is automatically added to any network operation where a time out condition is possible and needs to be handled. **Retry count** The maximum number of reconnects that will be attempted if a network connection is lost. **Retry Interval** The number of seconds between each reconnect attempt. **Frame interval** The frame interval is the number of transmission blocks between synchronization points.



Field	Description
Override the default TCP receive buffer and set to:	Specifies the TCP receive buffer size for TCP socket connections established from the UDM primary server to the UDM secondary server. The option has no effect for UDM two party transfers. The UDM manager and Universal Broker configuration affects UDM two party transfers. Refer to the Universal Broker tcp_recv_buffer option to set the TCP receive buffer for the other end of the socket connection. The socket TCP buffer is used by TCP to determine the size of the TCP window it advertises to the sending TCP. If both TCP implemenations support window scaling (as defined in RFC 1323), buffer sizes greater then 65535 will be used for the TCP window size. Note that both the sending and receiving TCP implemenations must support window scaling for it to be used in the TCP connection. Adjusting the TCP receive buffer size may improve bulk data transfer performance for certain types of transmission links when transfering data from the UDM secondary server to the UDM primary server. The format of the value is nnnnb, where 'nnnn' is a numeric value and 'b' is one of the following optional unit specifiers: • b - bytes (the default, used if no unit is specified) • k - kilobytes, where 1 kb = 1024 bytes • m - megabytes, where 1 mb = 1048576 bytes • g - gigabytes, where 1 gb = 1073741824 bytes Note that the maximum allowable value for this option is 1073725440 bytes, which can be requested using a value of 1g.
Override the default TCP send buffer and set to	Specifies the TCP send buffer size for TCP socket connections established from the UDM primary server to the UDM secondary server. The option has no effect for UDM two party transfers. The UDM manager and Universal Broker configuration affects UDM two party transfers. Refer to the Universal Broker tcp_send_buffer option to set the TCP send buffer for the other end of the socket connection. Adjusting the TCP send buffer size may improve bulk data transfer performance for certain types of transmission links when transfering data from the UDM manager to UDM server. The format of the value is nnnnb, where 'nnnn' is a numeric value and 'b' is one of the following optional unit specifiers: • b - bytes (the default, used if no unit is specified) • k - kilobytes, where 1 kb = 1024 bytes • m - megabytes, where 1 mb = 1048576 bytes • g - gigabytes, where 1 gb = 1073741824 bytes

	Note that the maximum allowable value for this option is 1073725440 bytes, which can be requested using a value of 1g.
Disable TCP packet coalescing	Select this option to disable the TCP packet coalescing algorithm (that is, Nagle algorithm). When this option is unchecked, the default TCP implementation is used, which may or may not disable the algorithm. Exercise caution when changing this option, as it could have a significant performance impact. See the tcp_no_delay configuration option in the UDM Reference Manual for more information. This option is checked by default.

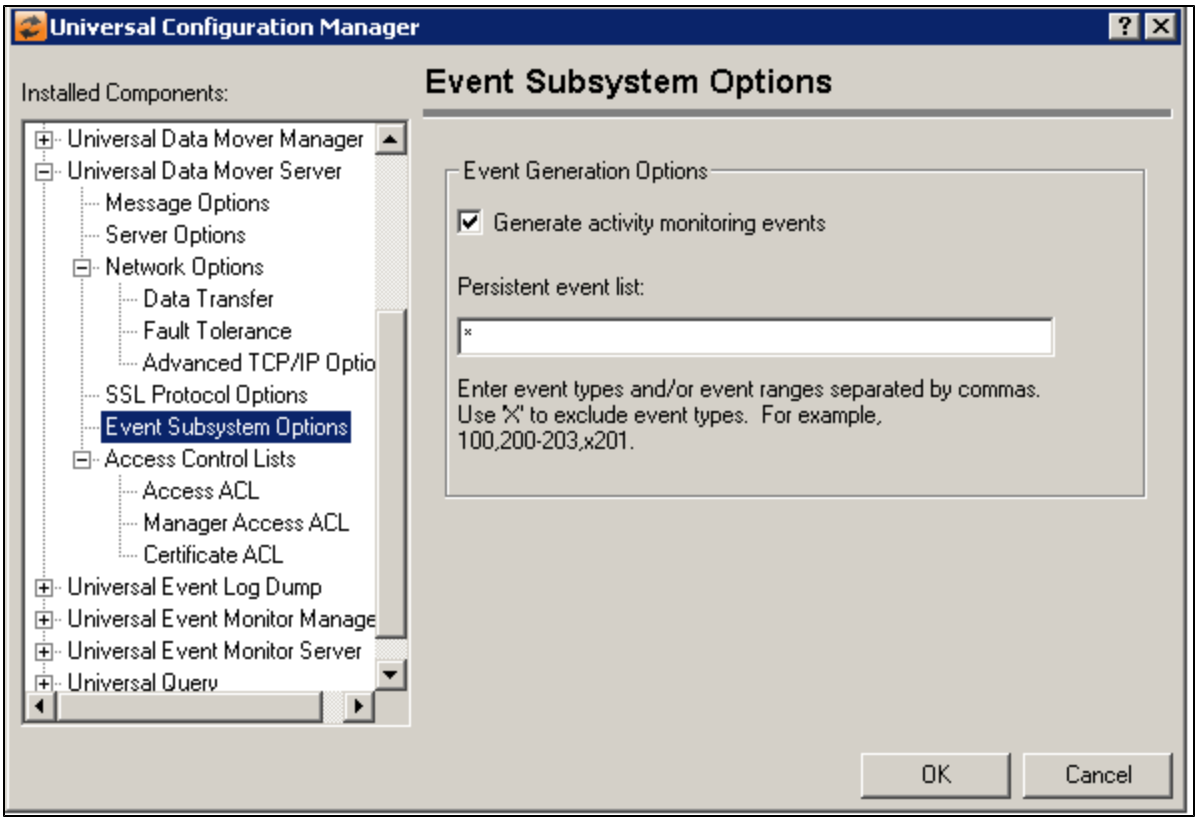
Universal Data Mover Server: SSL/TLS Protocol Options



Field	Description
Data Session SSL cipher List	Universal Data Mover Server uses the SSL/TLS protocol for transmission of data across the network. A data session is established within the application that provides the connections necessary for the transfer of data to take place. The SSL/TLS cipher suites available for communication across the data session are listed here. The Server will compare the Manager's cipher list to its own list of ciphers. The result of the comparison will be an agreed-upon cipher used for communication over the data session. Use the up/down arrows on the right to sort the list and establish a priority for the selected ciphers. The first one in the list that is accepted by the Server will be the one used. The following cipher suites are available: • RC4-SHA -- 128-bit RC4 encryption, SHA-1 message digest • RC4-MD5 -- 128-bit RC4 encryption, MD5 message digest • AES256-SHA -- 256-bit AES encryption, SHA-1 message digest • AES128-SHA -- 128-bit AES encryption, SHA-1 message digest • DES-CBC3-SHA -- 128-bit Triple-DES encryption, SHA-1 message digest • DES-CBC-SHA -- 128-bit DES encryption, SHA-1 message digest • NULL-SHA -- No encryption, SHA-1 message digest • NULL-MD5 -- No encryption, MD5 message digest • NULL-NULL -- Disables the SSL/TLS protocol
Require and	Select this option to require encryption over the control session between the UDM Manager and Server. When this option is selected, the UDM Server will not accept a control session SSL/TLS cipher of NULL-NULL from the remote UDM Manager.

encrypted control session	When this option is not selected, the UDM Server will accept any control session cipher from the UDM Manager, even one that disables SSL/TLS encryption. This option is checked by default.
Negotiate sessions using	(No help provided.)

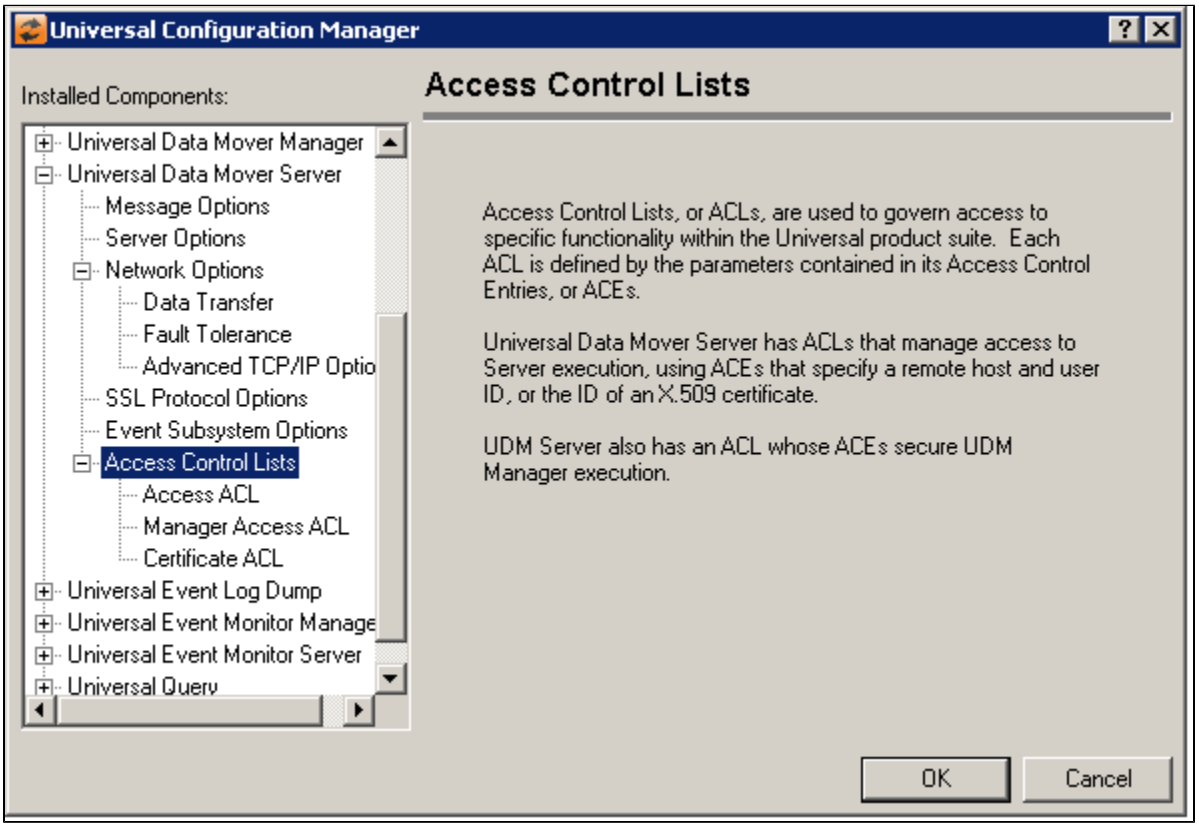
Universal Data Mover Server: Event Subsystem Options



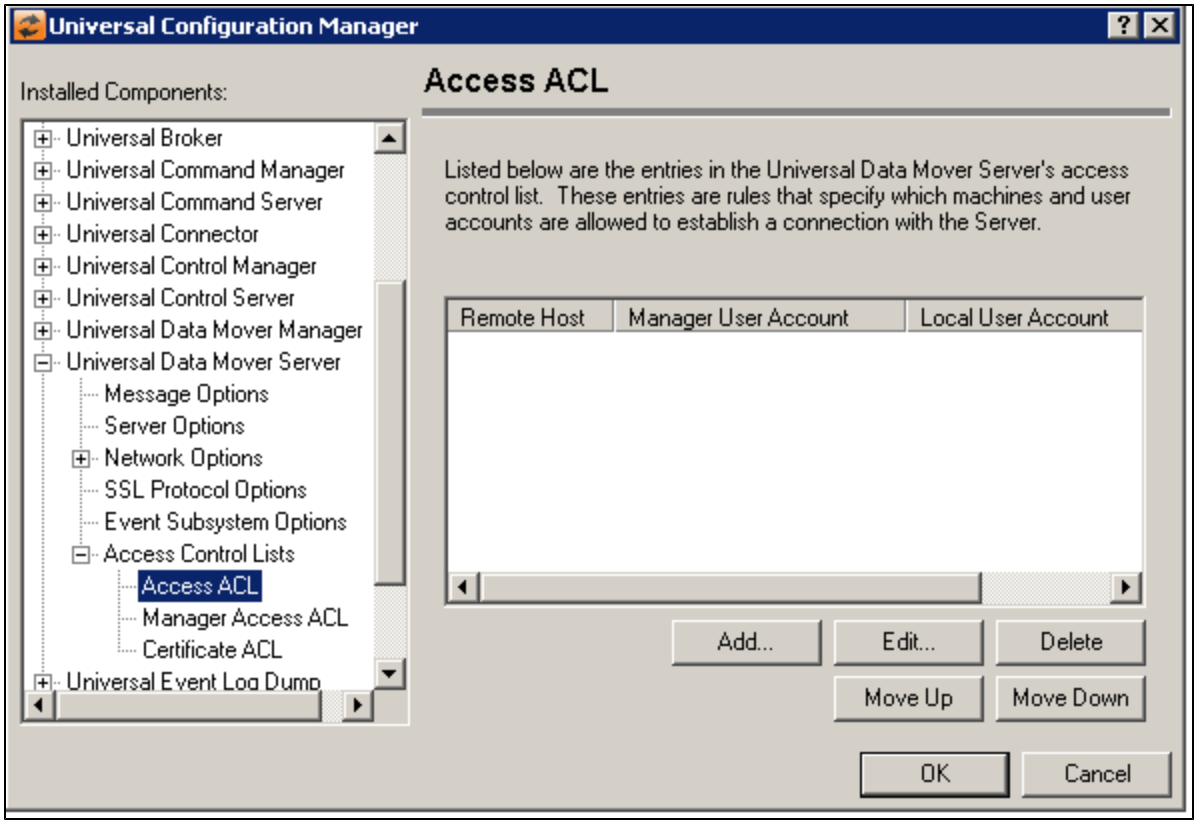
Field	Description
Event Generation Options	**Generate activity monitoring events** When this option is selected, the Universal Data Mover Serve will generate events that monitor product activity. This option is selected by default. **Persistent event list** Specifies which events are to be generated and processed as persistent events. A persistent event is saved in a Universal Enterprise Controller (UEC) event database for long-term storage. Enter a list and/or range of numeric event types, separated by a comma. To exclude an event or range of events, precede the entry with an upper- or lower-case 'X'. Use an asterisk ***** to specify all event types. For example: • x* Do not generate any events (this is the default) • * Generate all event types • 100 Generate all event types except 100 • 100,200-205 Generate event type 100 and 200 through 205 • 100,200-205,x202 Generate event types 100 and 200 thru 205, excluding event type 202

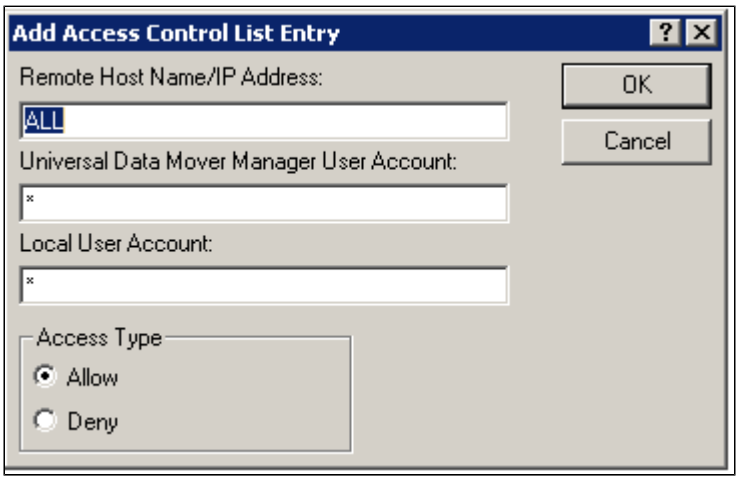
	Entries are processed from left to right, and the list is order dependent. For example, an entry of x*,100 prevents generation of all event types except 100, while an entry of 100,x* prevents generation of all event types (the right-most entry takes precedence).

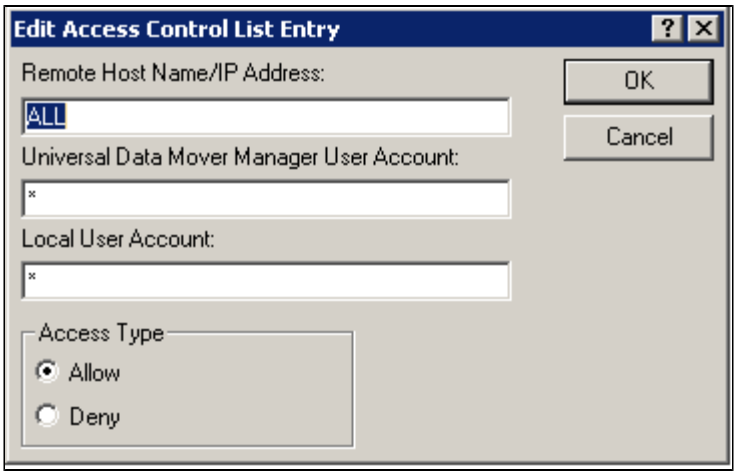
Universal Data Mover Server: Access Control Lists

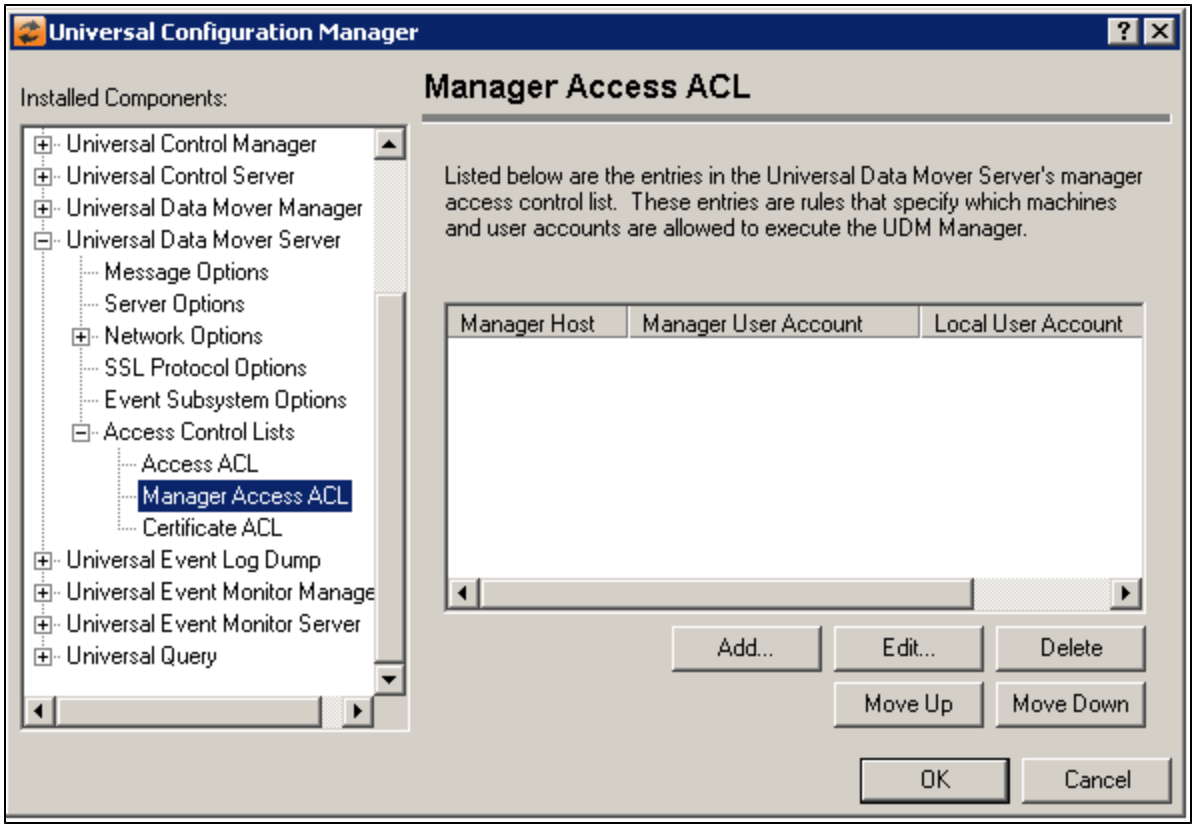


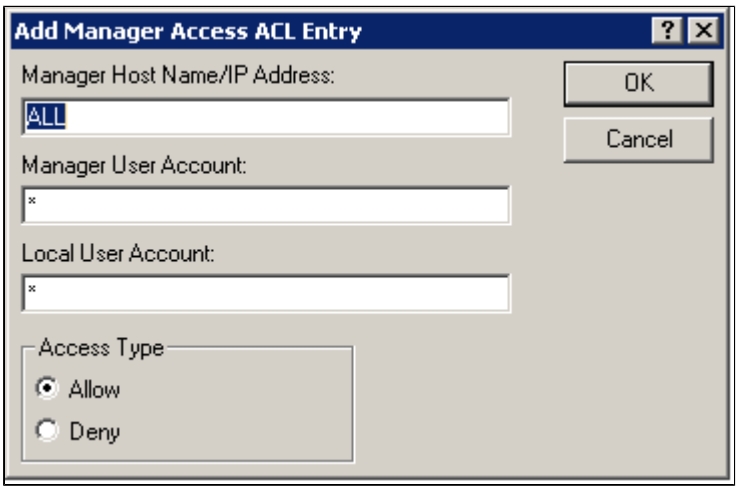
Universal Data Mover Server: Access Control Lists - Access ACL



Field	Description
Access Control List	Displays the entries contained in the Universal Data Mover Server Access Control List (ACL). This list is used to grant or deny access to Universal Data Mover Server based on the remote system's IP address, the user account running the Universal Data Mover Manager, and the user account with which the Server is executed. The order in which ACL entries are evaluated and applied are based on their position in this list. Use the Move Up and Move Down buttons to change an entry's priority.
Add Button	Displays a dialog box that allows an entry to be added to the Access Control List. The fields in this dialog are populated with default values.  **Remote Host Name/IP Address** The IP address or name of the remote host for which access will be allowed or denied. To apply an access control entry to all IP addresses, use the keyword ALL (in all caps). **Universal Data Mover Manager User Account**

	(No help available.) **Local User Account** The local user account that will be used to execute the file transfer. **Access Type** Will cause the Universal Data Mover Server to accept/reject incoming connections from the specified host, provided the Universal Data Mover Manager is executed by the specified user account and the process is owned by the specified local user account.
Edit Button	Displays a dialog that allows the selected Access Control List entry to be modified.  **Remote Host Name/IP Address** The IP address or name of the remote host for which access will be allowed or denied. To apply an access control entry to all IP addresses, use the keyword ALL (in all caps). **Universal Control Manager User Account** (No help available.) **Local User Account** The local user account that will be used to execute the file transfer. **Access Type** Will cause the Universal Data Mover Server to accept/decline incoming connections from the specified host, provided the Universal Data Mover Manager is executed by the specified user account and the process is owned by the specified local user account.
Delete Button	Removes the selected item from the Access Control List.
Move Up Button	Moves the selected item up one position in the list.
Move Down Button	Moves the selected item down one position in the list.



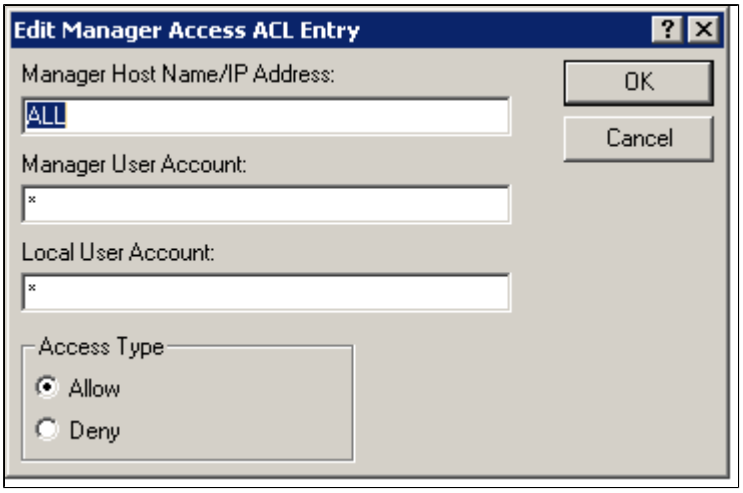
Field	Description
Manager Access ACL List	Displays the entries contained in the UDM Server Manager Access Control List (ACL). The order in which ACL entries are evaluated and applied are based on their position in this list. Use the Move Up and Move Down buttons to change an entry's priority.
Add Button	

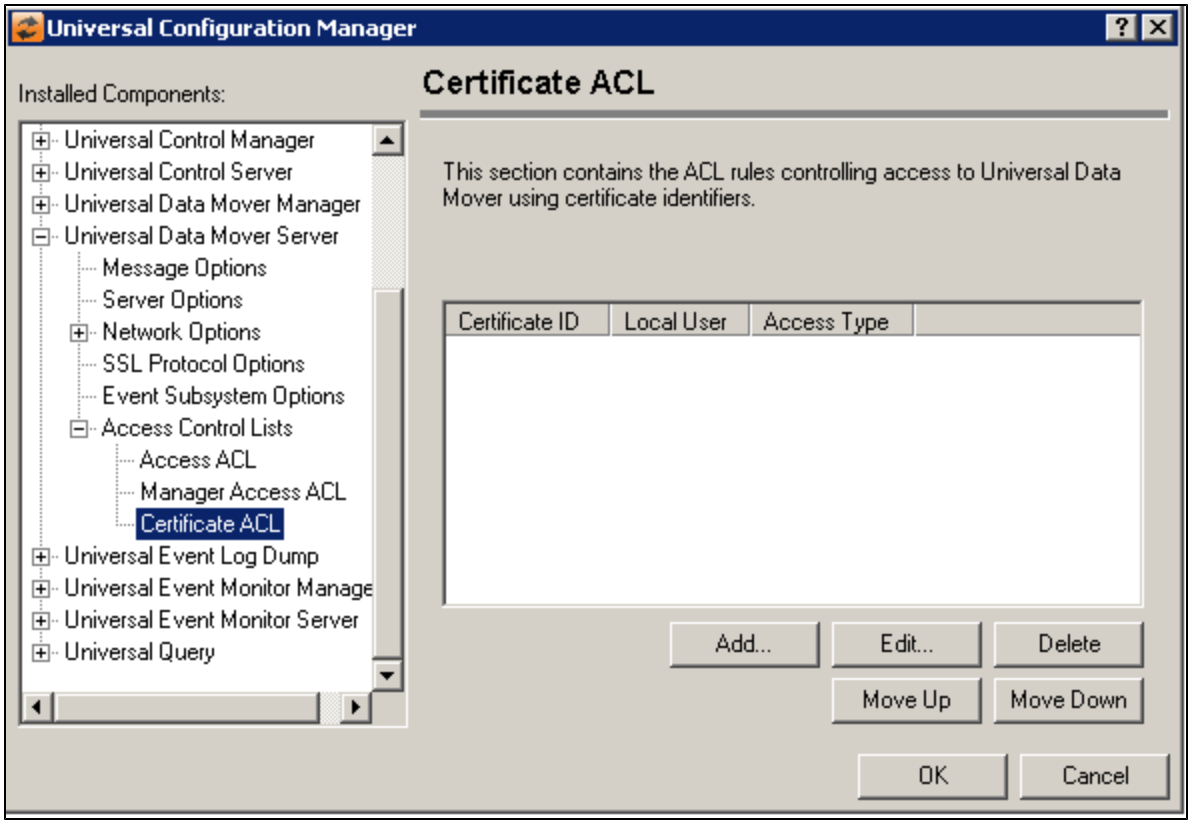
Manager Host Name / IP Address
The IP address or name of the UDM Manager host for which access will be allowed or denied. To apply an access control entry to all IP addresses, use the keyword ALL (in all caps).

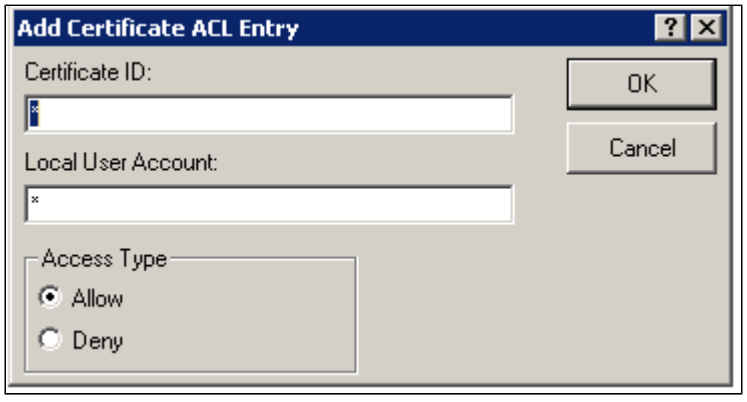
Manager User Account
The ID of the user account executing the Universal Data Mover Manager.

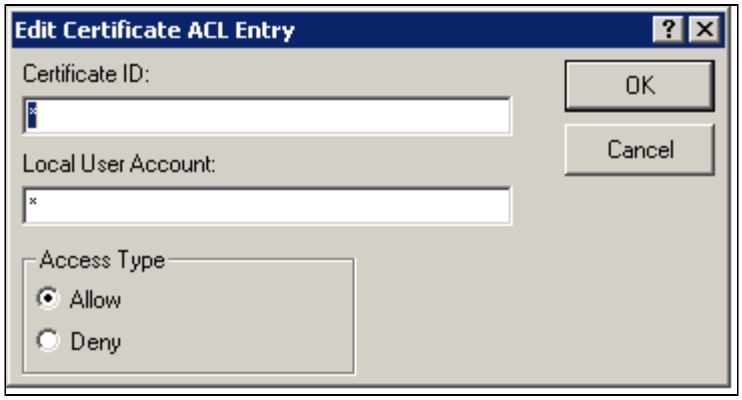
Local User Account
The local user account that will be used to execute the file transfer.

Access Type
Will permit/reject file transfers by UDM Managers executed on the specified host, by the specified user account, for files transferred to the target system using the specified local user account.

Edit Button	 **Manager Host Name / IP Address** The IP address or name of the UDM Manager host for which access will be allowed or denied. To apply an access control entry to all IP addresses, use the keyword ALL (in all caps). **Manager User Account** The ID of the user account executing the Universal Data Mover Manager. **Local User Account** The local user account that will be used to execute the file transfer. **Access Type** Will permit/reject file transfers by UDM Managers executed on the specified host, by the specified user account, for files transferred to the target system using the specified local user account.
Delete button	Removes the selected item from the List.
Move Up button	Moves the selected item up one position in the list.
Move Down button	Moves the selected item down one position in the list.



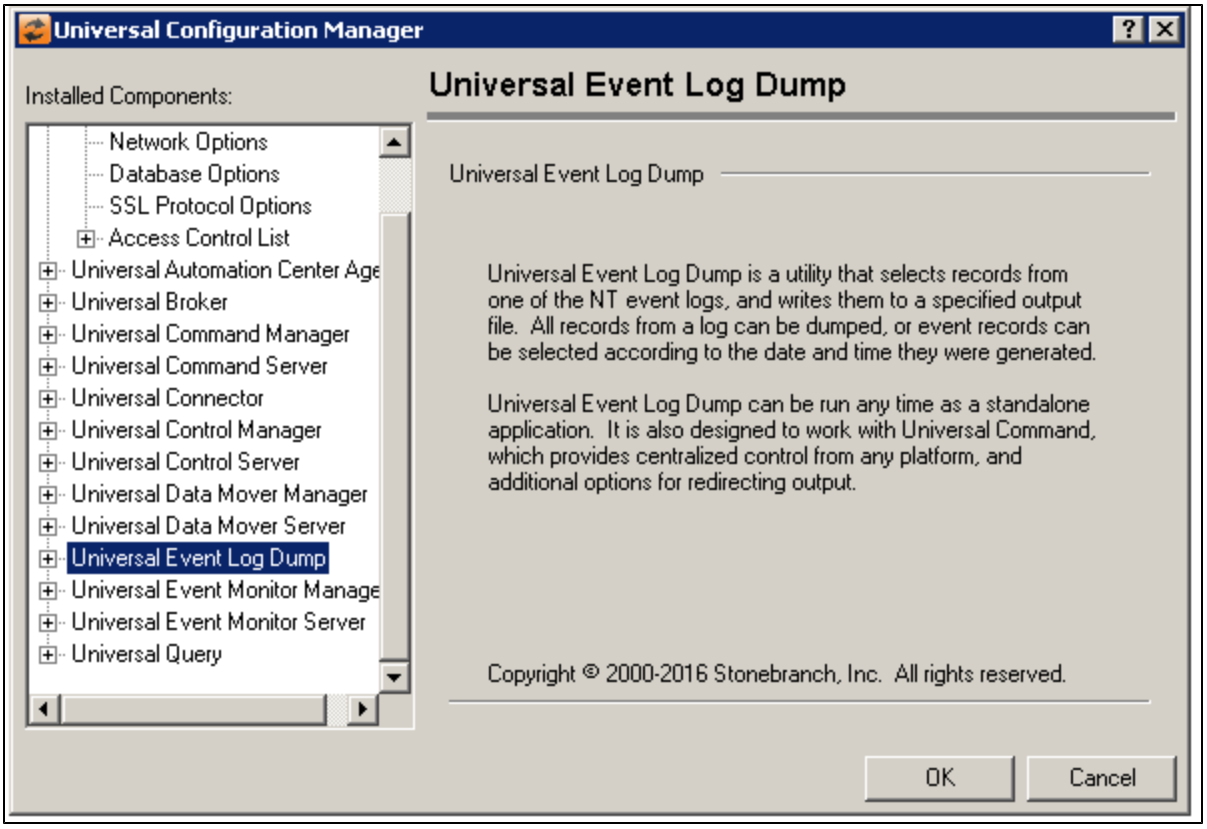
Field	Description
Certificate ACL List	Displays the entries contained in the Universal Data Mover Certificate Access Control List (ACL). The order in which ACL entries are evaluated and applied are based on their position in this list. Use the Move Up and Move Down buttons to change an entry's priority.
Add Button	Displays a dialog box that allows an entry to be added to the Access Control List. The fields in this dialog are populated with default values.  **Certificate ID** The ID of a certificate that has a corresponding entry in the Universal Broker's certificate map. **Local User Account** The local user account that will be used to execute the file transfer. **Access Type** Will Allow/Deny file transfers when a certificate with the specified ID is presented to the UDM Server and the transfer is executed by the specified local user account.
Edit Button	Displays a dialog that allows the selected Access Control List entry to be modified.

	 **Certificate ID** The ID of a certificate that has a corresponding entry in the Universal Broker's certificate map. **Local User Account** The local user account that will be used to execute the file transfer. **Access Type** Will Allow/Deny file transfers when a certificate with the specified ID is presented to the UDM Server and the transfer is executed by the specified local user account.
Delete Button	Removes the selected item from the Access Control List.
Move Up Button	Moves the selected item up one position in the list.
Move Down Button	Moves the selected item down one position in the list.

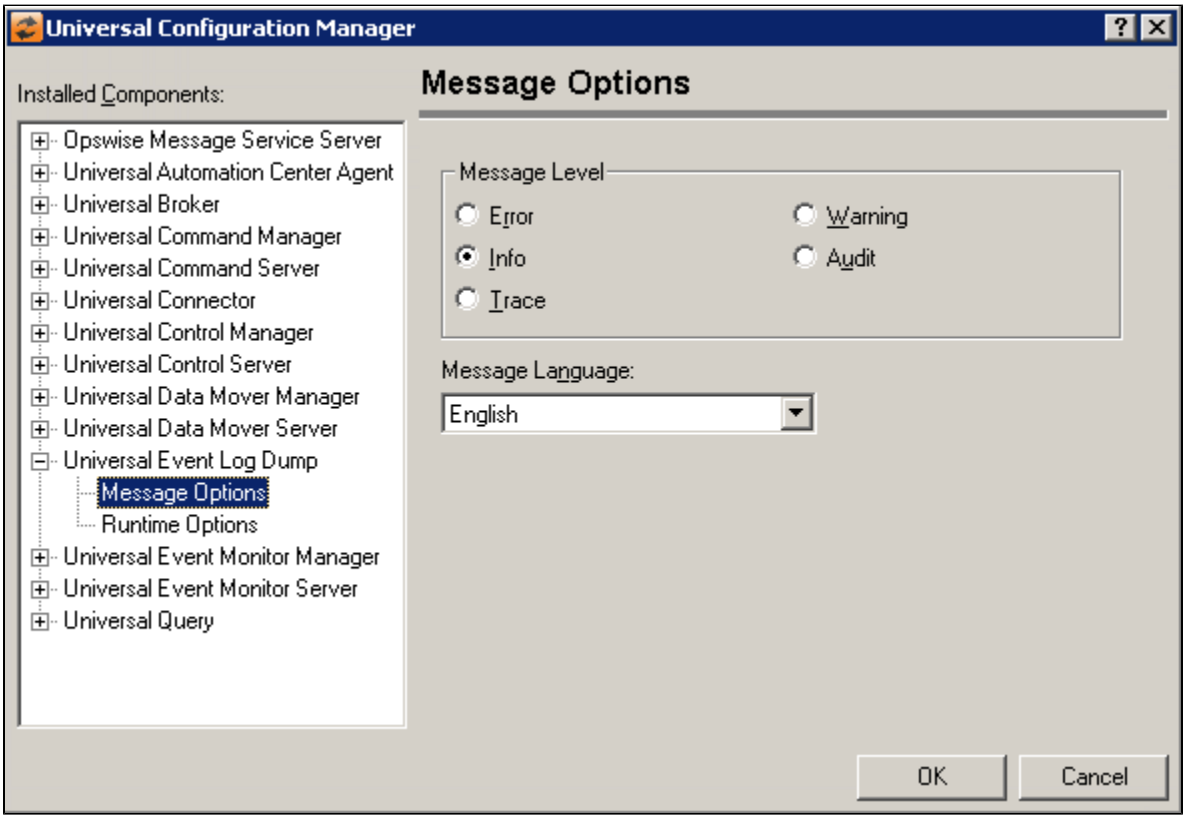
Universal Configuration Manager - Universal Event Log Dump

- Universal Event Log Dump
- Universal Event Log Dump - Message Options
- Universal Event Log Dump: Runtime Options

Universal Event Log Dump



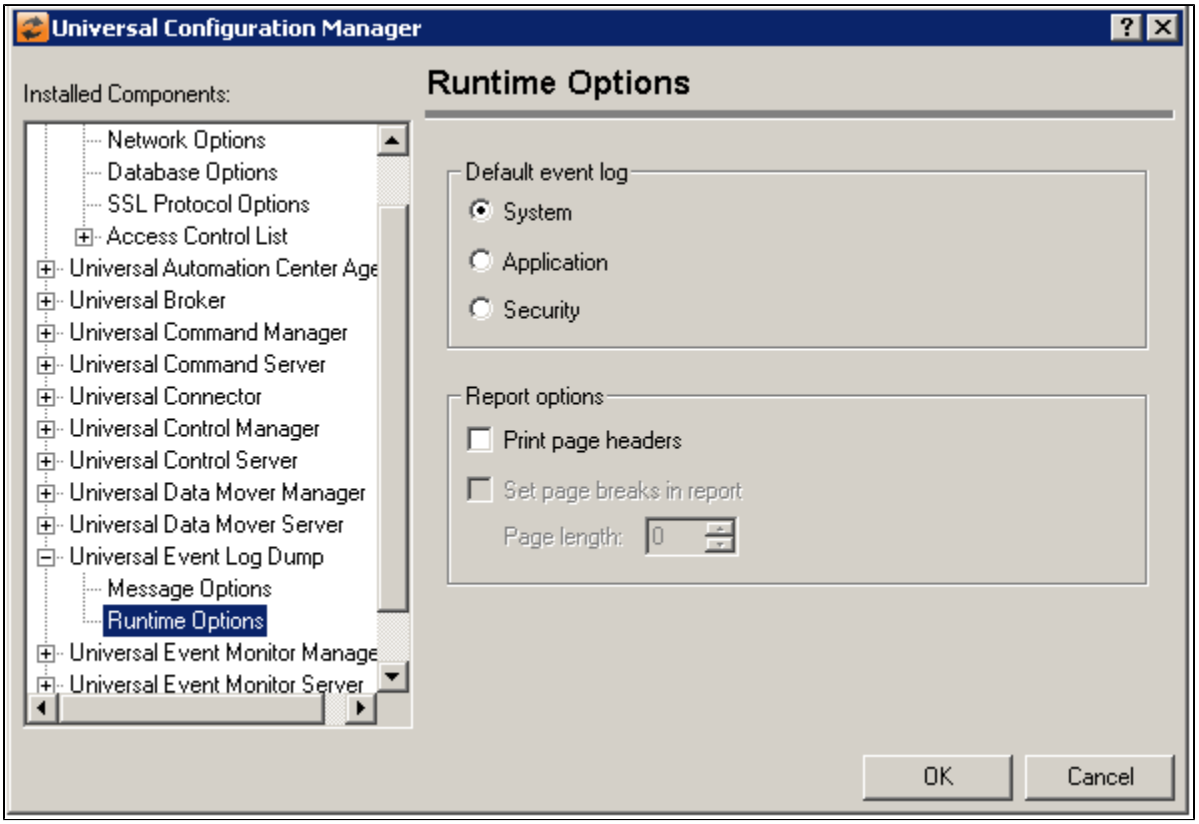
Universal Event Log Dump - Message Options



Field	Description
Message Level	**Error** Turns on message logging for errors only.

	Warning Turns on message logging for warnings and errors. **Info** Turns on message logging for all informational, warning and error messages. **Audit** Turns on message logging for all audit, informational, warning and error messages. **Trace** Allows all messages to be logged, including those used for debugging purposes.

Universal Event Log Dump: Runtime Options



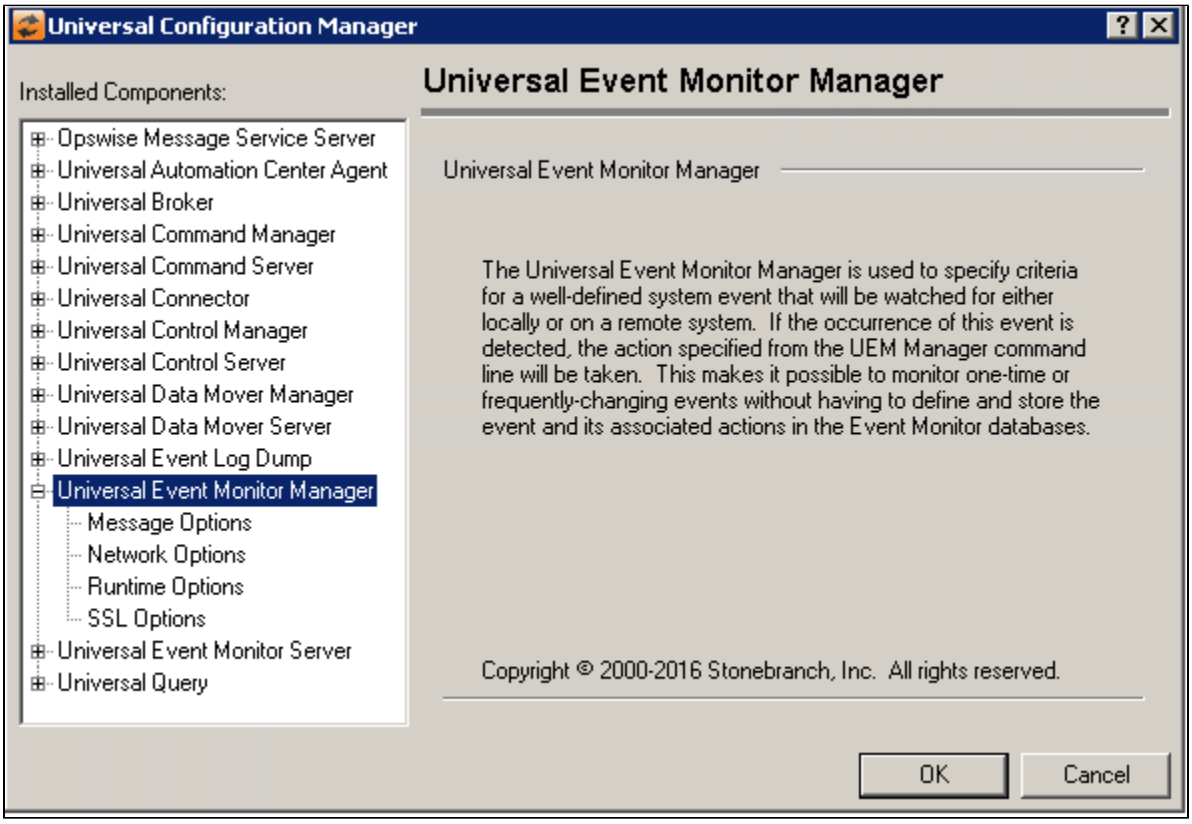
Field	Description
Default Event Log	**System** Sets the system log as the default event log. **Application** Sets the application log as the default event log. **Security** Sets the security log as the default event log.
Report Options	**Print page header** Specifies whether or not column headings should be printed at the top of each page. Checking this box enables the page break options. **Set page break in report** Specifies whether or not the report should be broken into "logical" pages, with column headings at the top of each page. Checking this box enables the page length edit control, which is used to specify the number of lines per page. **Page Length** Specifies the number of lines to print on each "logical" page of the event log dump report. When this number is reached, column headings are printed. If page headers are not being printed, this value is ignored.

Universal Configuration Manager - Universal Event Monitor Manager

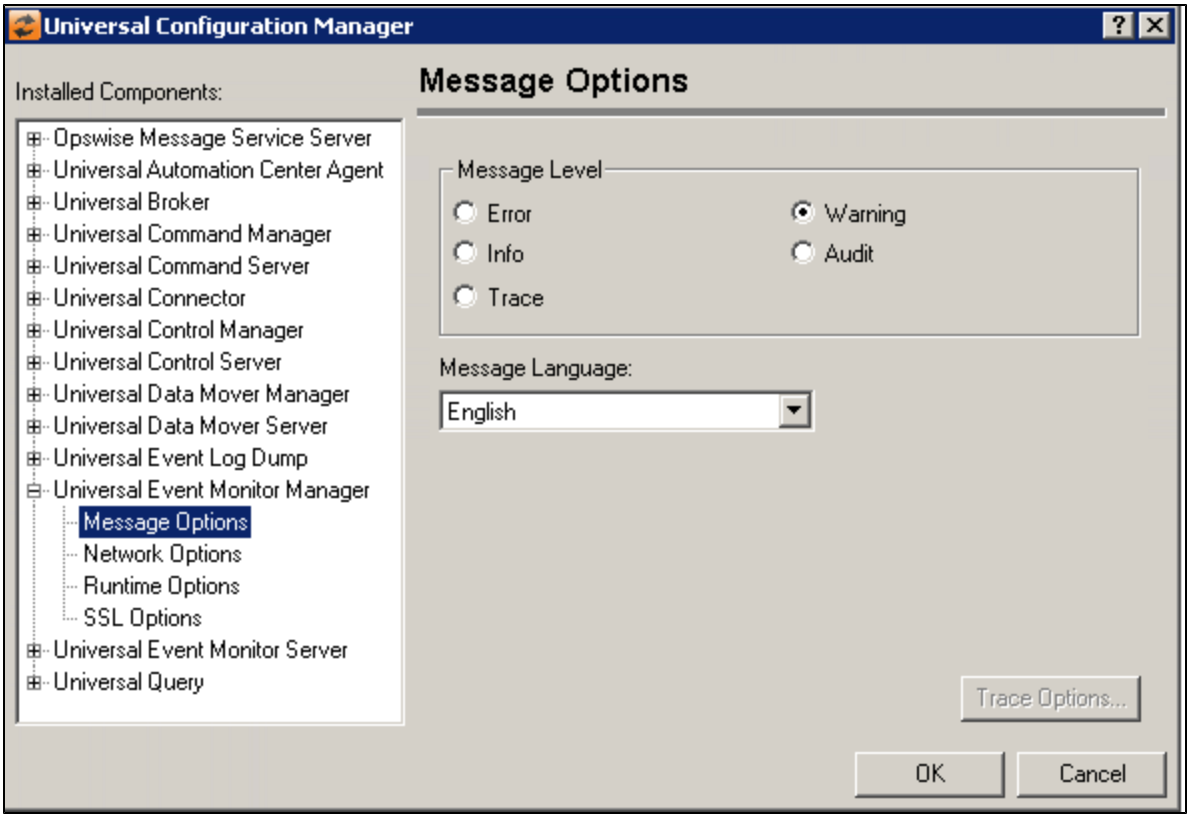
- Universal Event Monitor Manager

- Universal Event Monitor Manager: Message Options
- Universal Event Monitor Manager: Network Options
- Universal Event Monitor Manager: Runtime Options
- Universal Event Monitor Manager: SSL/TLS Options

Universal Event Monitor Manager

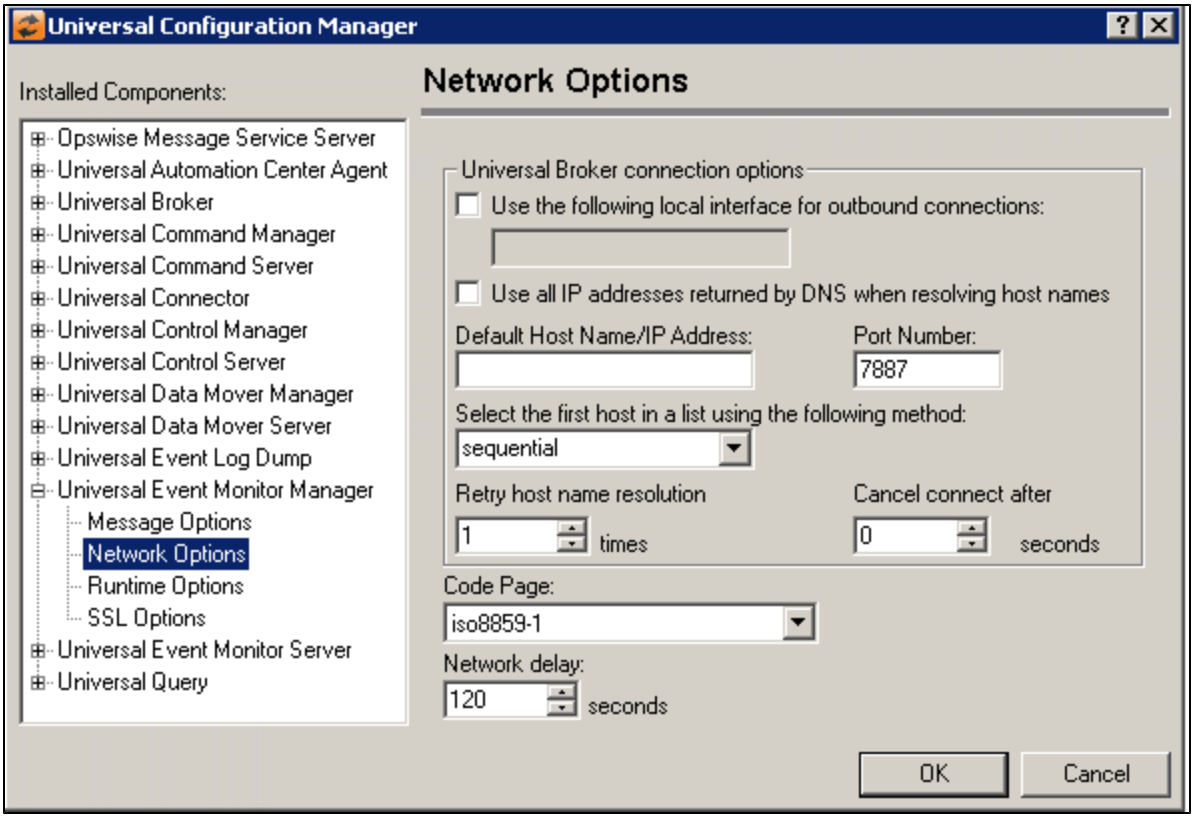


Universal Event Monitor Manager: Message Options



Field	Description
Message Level	**Error** Turns on message logging for errors only. **Warning** Turns on message logging for warnings and errors. **Info** Turns on message logging for all informational, warning and error messages. **Audit** Turns on message logging for all audit, informational, warning and error messages. **Trace** Allows all messages to be logged, including those used for debugging purposes.
Message Language	Specifies the language used to display error messages.

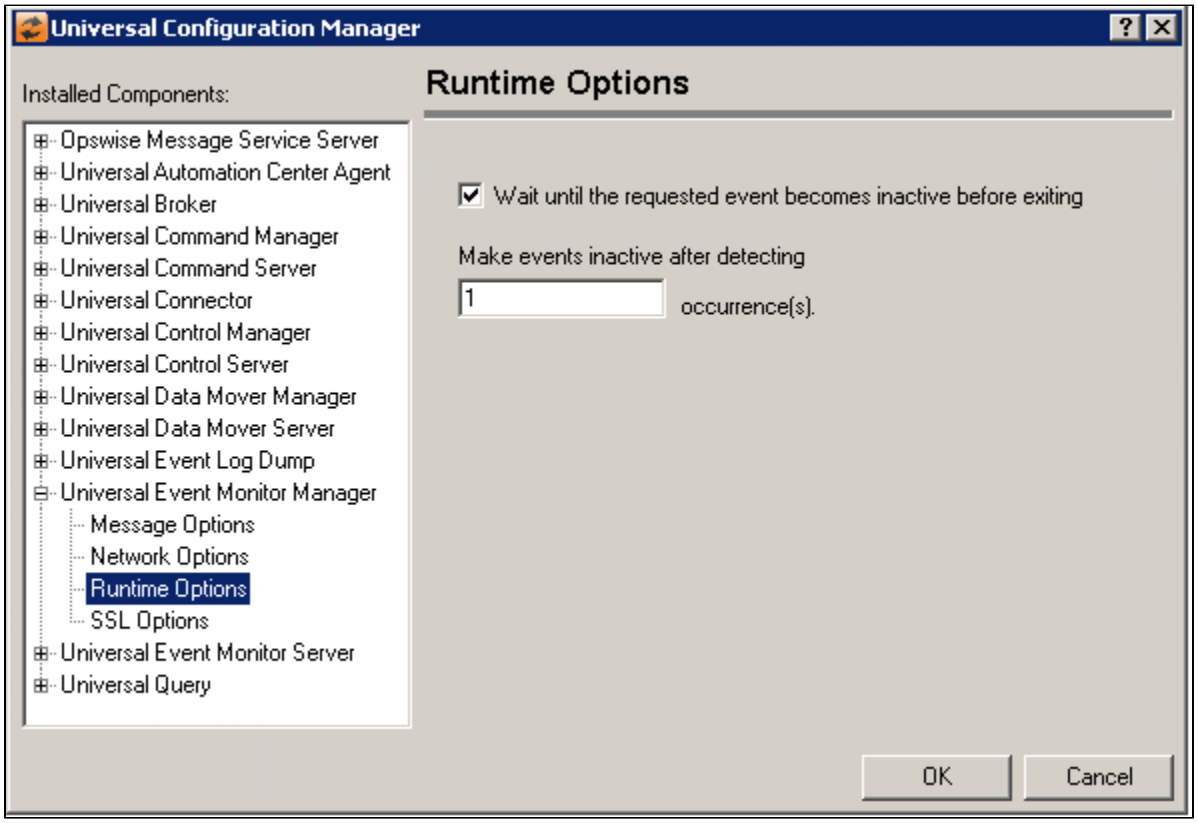
Universal Event Monitor Manager: Network Options



Field	Description
Universal Broker connection options	**Use the following local interface for outbound connections** When this option is selected, the IP address or alias of a local interface may be specified that will be used whenever UEM Manager connects to a Universal Broker. If this option is not selected, any available local interface will be used. When more than one local interface is available, this option may be used to specify the IP address or host name that should be used whenever UEM Manager connects to a Universal Broker. To have UEM Manager use any local interface available, leave this value blank. **Use all IP addresses returned by DNS when resolving host names** When this option is selected, all IP addresses returned by a DNS query for a given host name are expanded in the list of hosts. When this option is not selected, only the first IP address for the resolved host name is used. This option is not selected by default. **Default Host Name/IP Address** Enter one or more valid IP addresses and/or host names – separated by a comma – to use as the default system(s) for event monitoring. A specific port number may also be specified for any one of the hosts in the list. The format for this list of one or more hosts is: `host1[port1,host2port2,...,hostnportn]` To monitor an event on the local system, enter a host name of localhost, or an IP address of 127.0.0.1. **Port Number** Enter a port number or service name that will be used to connect to Universal Broker. **Select the first host in a list using the following method** This option specifies how entries within a list of hosts are accessed as the Universal Event Monitor Manager attempts to establish a connection with a Universal Broker. When the sequential selection method is chosen, connection attempts begin with the first host in the list. Connection attempts continue with each subsequent host in the list, until a connection to a Universal Broker is established or until all hosts in the list have been tried. When the random method is chosen, connection attempts begin with a randomly selected host. Connection attempts continue with each host in the list until a connection is established or until the first host selected is reached again. The default value for this option is sequential. **Retry host name resolution** Indicates the number of attempts that will be made to resolve a host name specified with the -host option on the Universal Event Monitor command line. This parameter can help resolve intermittent "host not found" errors. **Cancel Connect After**

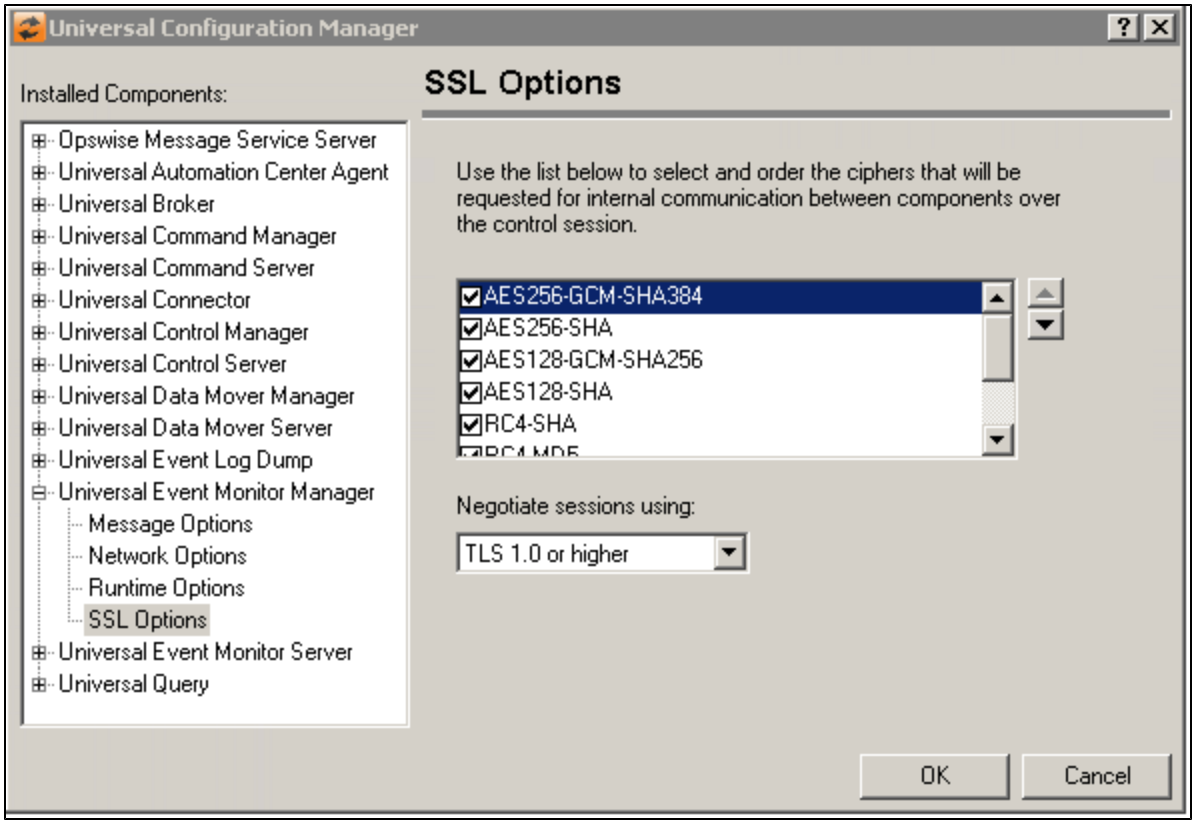
	This option specifies how long the Universal Event Monitor Manager will wait for each connection attempt to a Universal Broker to complete before moving on to the next host in the list. If there are no more hosts in the list, or connection attempts to all the hosts time out, the Manager will terminate. The default value for this option is 0 (zero), which instructs the UEM Manager to use the default TCP/IP timeout value. This results in a blocking connect attempt, and is provided to preserve compatibility with pre-v320 behavior.
Code Page	Sets the translation table used to translate text-based data transmitted across the network.
Network Delay	The network delay value specifies the maximum acceptable delay in the network. Should it take longer than the specified delay time to send or receive data over the network, the operation will end with a time out error. The default is 120 seconds.

Universal Event Monitor Manager: Runtime Options



Field	Description
Wait until the requested event becomes inactive before exiting	When this option is checked, the UEM Manager will, by default, wait for the event being monitored to become inactive before completing. This means that the Manager will not end until the demand-driven UEM Server that was started to monitor the event ends. To cause the Manager to return as soon as the UEM Server begins monitoring the event, leave this option unchecked.
Make events inactive after detecting	This option is used to track the number of occurrences that are detected by a demand-driven UEM Server (that is, one started from a UEM Manager) for a given event. When the Server has detected the specified number of occurrences, the event will become inactive even if the inactive date/time has not elapsed. If the specified count is not reached by the inactive date/time, the event will become inactive at that time. A value of 0 (zero) will effectively disable this feature.

Universal Event Monitor Manager: SSL/TLS Options

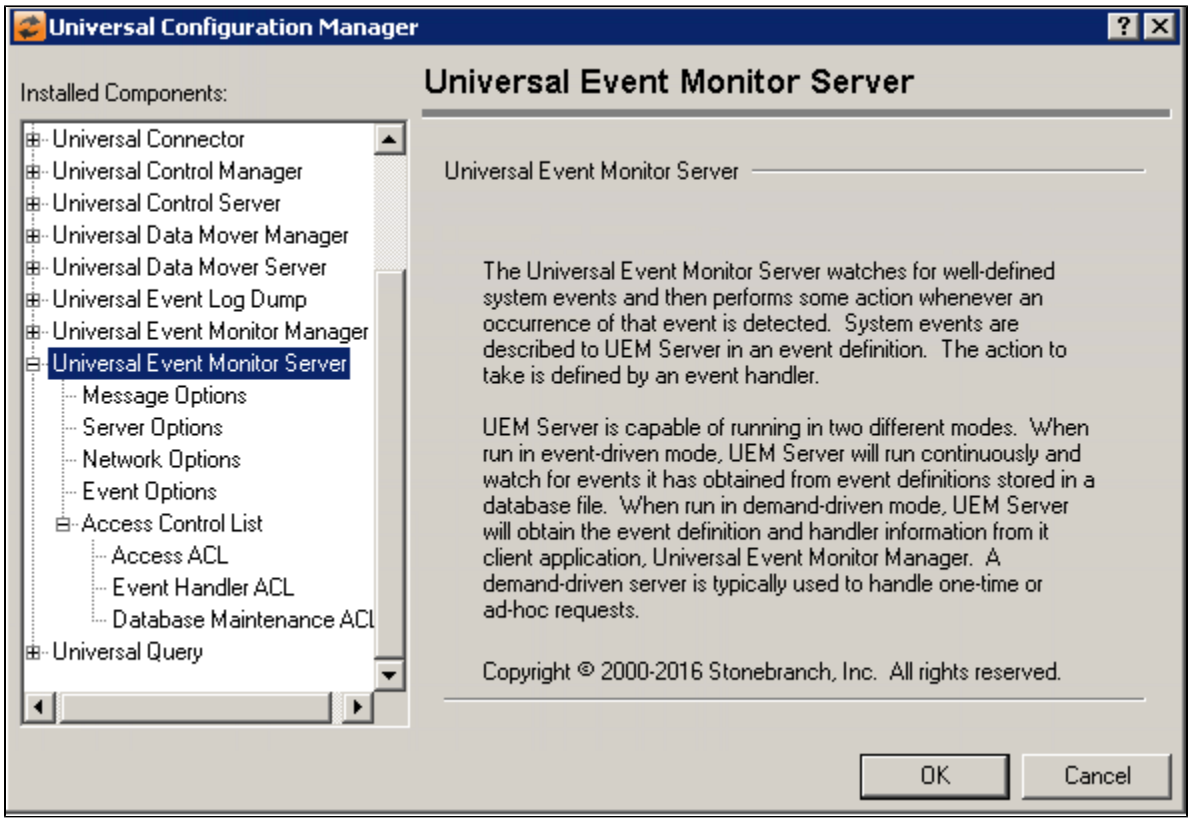


Field	Description
Internal Communication Ciphers List	Universal Event Monitor uses the SSL/TLS protocol for internal communication between application components. A control session is established within the application that provides the connections necessary for this communication to take place. The SSL/TLS cipher suites available for communication across the control session are listed here. The selected cipher suites will be forwarded to the Universal Event Monitor Server. The Server will compare this list to its own list of ciphers it is capable of accepting. The result of the comparison will be an agreed-upon cipher used for communication over the control session. If you are unsure which ciphers will be accepted by the Server, it is best to leave them all selected. Use the up/down arrows on the right to sort the list and establish a priority for the selected ciphers. The first one in the list that is accepted by the Server will be the one used. The following cipher suites are available: • RC4-SHA -- 128-bit RC4 encryption, SHA-1 message digest • RC4-MD5 -- 128-bit RC4 encryption, MD5 message digest • AES256-SHA -- 256-bit AES encryption, SHA-1 message digest • AES128-SHA -- 128-bit AES encryption, SHA-1 message digest • DES-CBC3-SHA -- 128-bit Triple-DES encryption, SHA-1 message digest • DES-CBC-SHA -- 128-bit DES encryption, SHA-1 message digest
Negotiate sessions using	(No help available.)

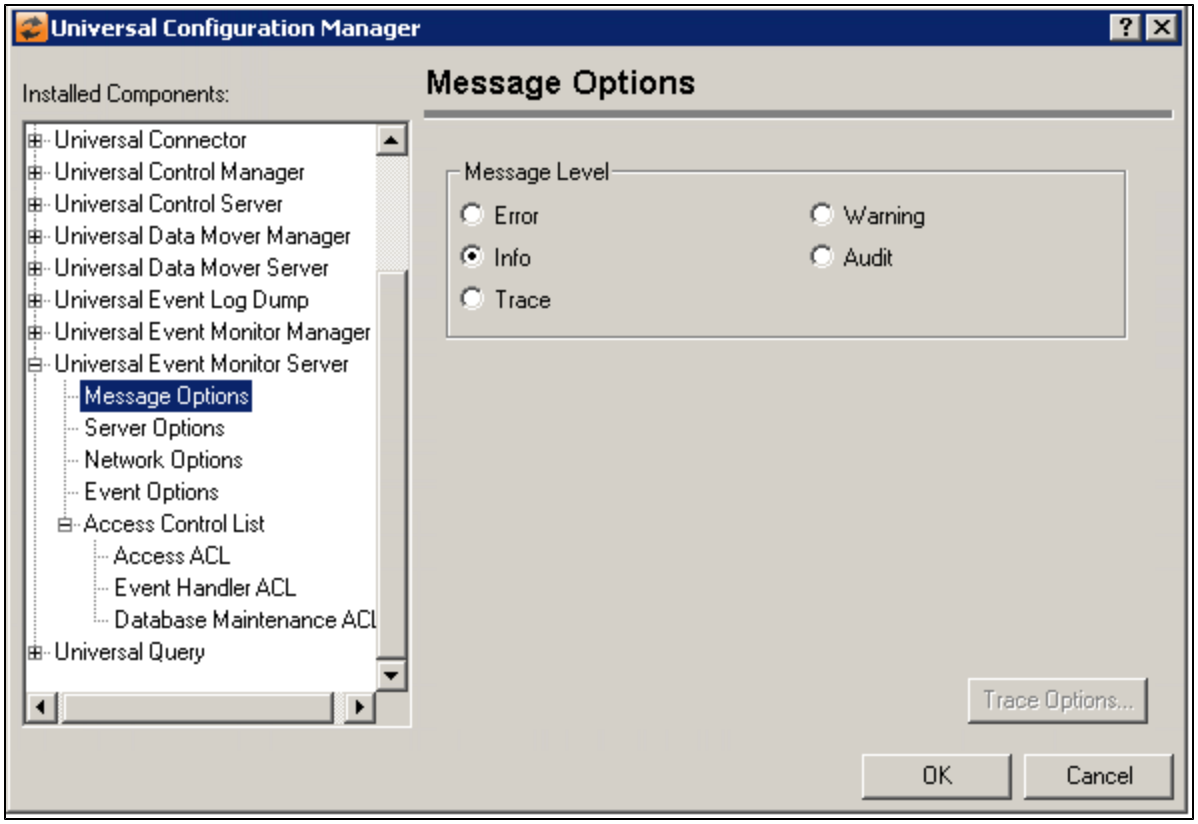
Universal Configuration Manager - Universal Event Monitor Server

- Universal Event Monitor Server
- Universal Event Monitor Server: Message Options
- Universal Event Monitor Server: Server Options
- Universal Event Monitor Server: Network Options
- Universal Event Monitor Server: Event Options
- Universal Event Monitor Server: Access Control List
  - Universal Event Monitor Server: Access Control List: Access ACL
  - Universal Event Monitor Server: Access Control List - Event Handler ACL
  - Universal Event Monitor Server: Access Control List - Database Maintenance ACL

Universal Event Monitor Server



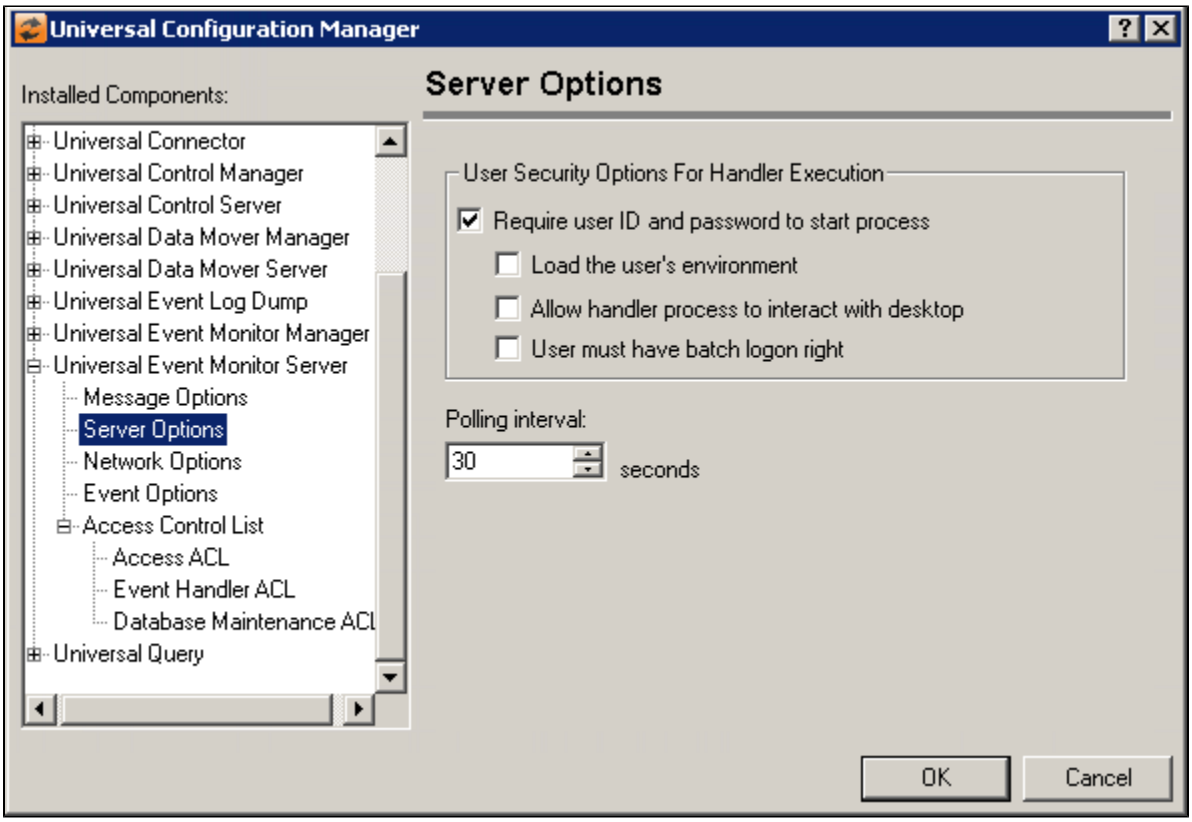
Universal Event Monitor Server: Message Options



Field	Description

Message Level	**Error** Turns on message logging for errors only. **Warning** Turns on message logging for warnings and errors. **Info** Turns on message logging for all informational, warning and error messages. **Audit** Turns on message logging for all audit, informational, warning and error messages. **Trace** Allows all messages to be logged, including those used for debugging purposes.

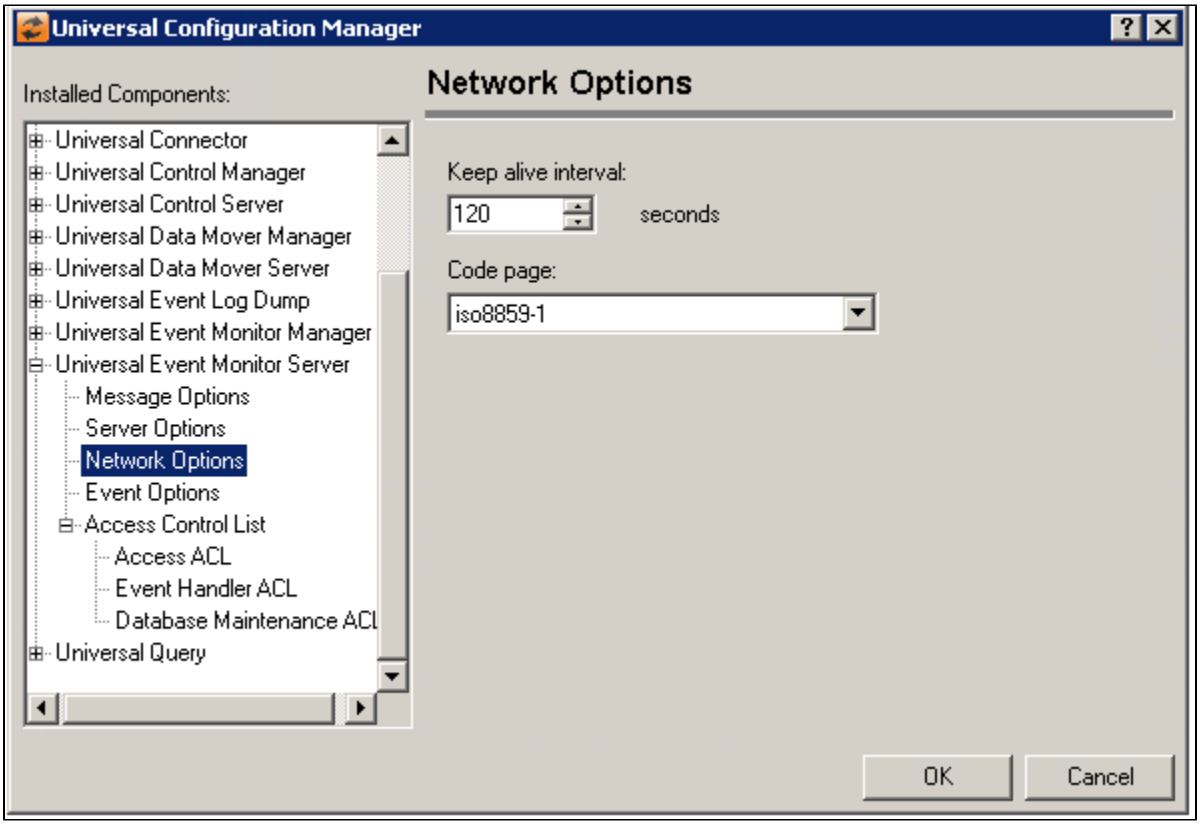
Universal Event Monitor Server: Server Options



Field	Description
User Security Options for handling Execution	**Require user ID and password to start process** When this option is checked, Universal Event Monitor will require a valid user ID and password to execute an event handler process. That process will then be executed in the security context of the specified user account. **Load the user's environment** When checked, causes Universal Event Monitor Server to load the profile and the environment settings for the user account specified in the event handler when the handler process is executed. This is necessary only if the process requires access to user-specific environment variables, or registry values that are only stored under the HKEY_CURRENT_USER key. **Allow handler process to interact with desktop** When this option is checked, event handler processes are run in a context that permits the current console logon session to interact with them. This also allows the process to invoke system functions that access desktop elements (for example, Windows, menus, buttons) associated with that session. Accordingly, this exposes a risk whereby a process could hijack the desktop and invoke malicious code using a (potentially) elevated security context. Leaving this option unchecked causes event handler processes to execute in a context that isolates them from the current interactive logon session. Unless event handler processes that require user interaction are executed, this is the recommended setting. This option only applies if a user account (in whose security context an event handler process is executed) is authenticated via

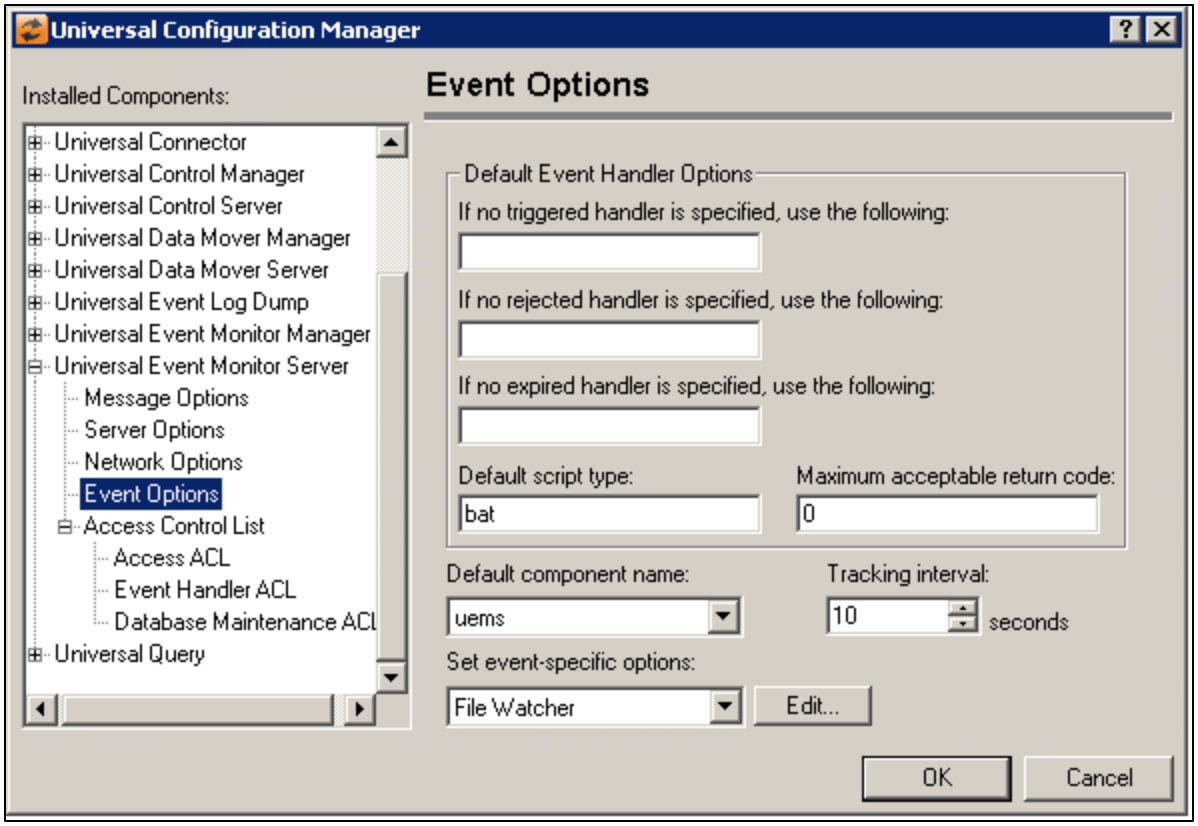
	an interactive logon. If the User must have batch logon right option is checked, a batch logon is done to establish the process's security context. This logon method requires additional privileges but disallows all interaction with the desktop. **User must have Batch logon right** When this option is checked, the Log on as a batch job user right must be granted to the user ID specified in the event handler before the process will run. This is an advanced user right that must be set using the Windows User Manager.
Polling Interval	Specifies the frequency with which UEM Server will check (and update, if necessary) the active or inactive state of an enabled event definition. For those events that are active, any new occurrences of system events will also be detected at this time.

Universal Event Monitor Server: Network Options

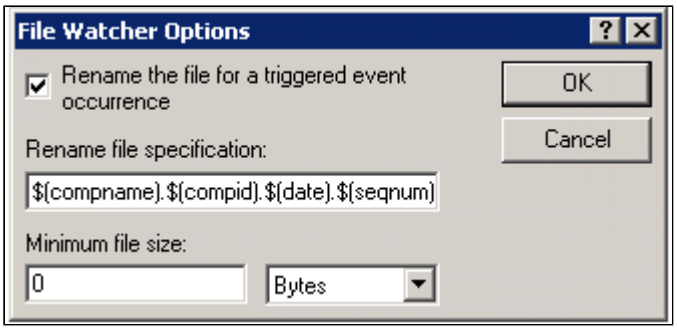


Field	Description
Keep alive interval	This value specifies the frequency with which a demand driven UEM Server sends a keep alive message to the Universal Event Monitor Manager. This message ensures that the control session between the UEM Manager and Server remains open even when there is no monitoring activity or other information to send across that connection. The default value for this option is 120 seconds. The maximum allowed value is 900 seconds. Set this value to 0 (zero) to instruct UEM Server not to send the keep alive message.
Code Page	Sets the translation table used to translate text-based data transmitted across the network.

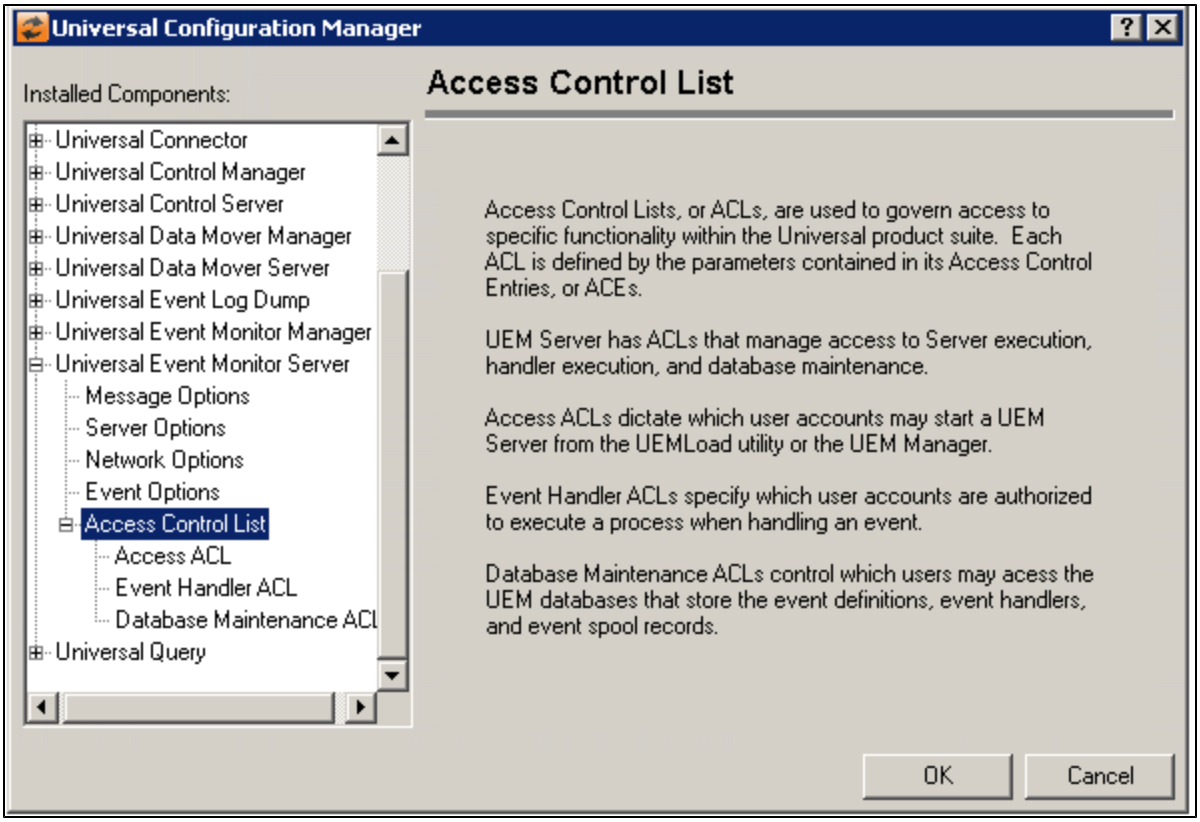
Universal Event Monitor Server: Event Options



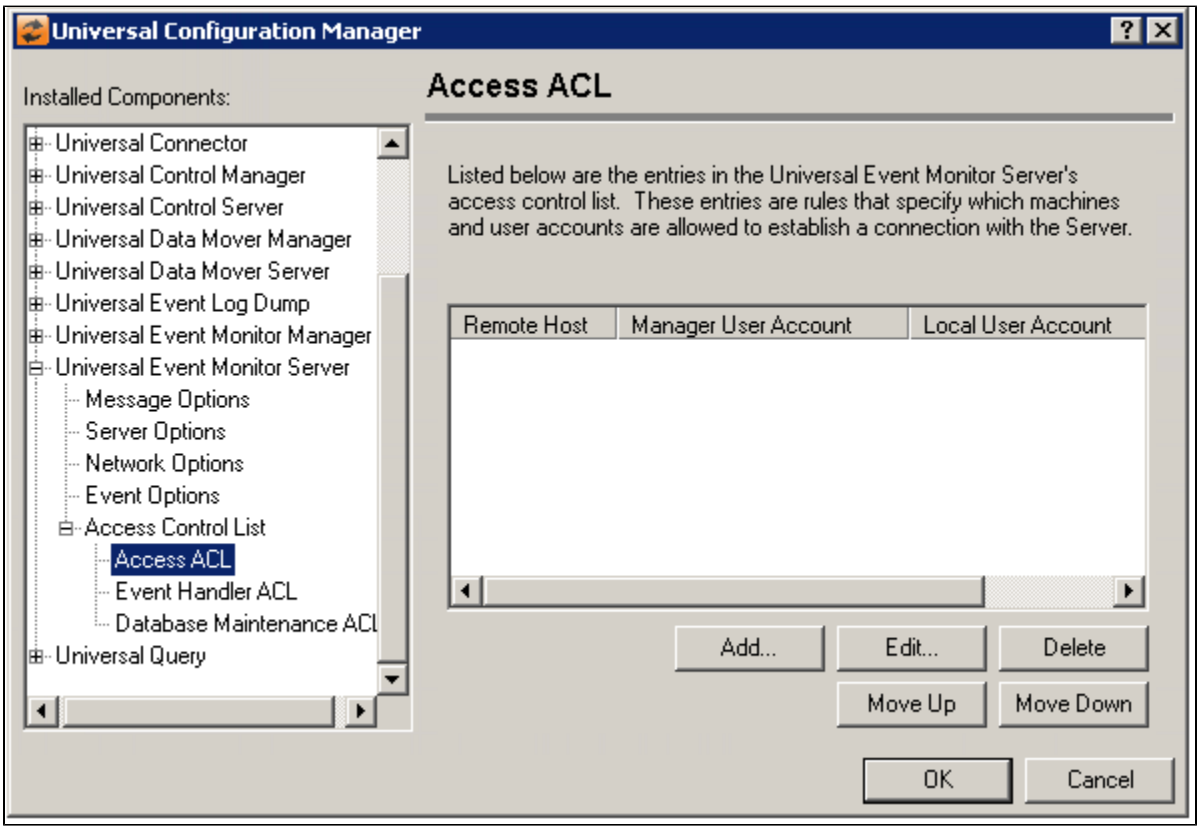
Field	Description
Default Event Handler Options	**If no triggered handler is specified, use the following** The ID of a stored event handler that will be executed when an occurrence of an event is triggered, if no such handler is specified in that event's definition. This value may be left blank to indicate that no action should be taken by default for triggered event occurrences. **If no reject handler is specified, use the following** The ID of a stored event handler that will be executed when an occurrence of an event is rejected, if no such handler is specified in that event's definition. This value may be left blank to indicate that no action should be taken by default for rejected event occurrences. **If no expired handler is specified, use the following** The ID of a stored event handler that will be executed when an an event expires, if no such handler is specified in that event's definition. This value may be left blank to indicate that no action should be taken by default for expired events. **Default script type** Specifies the how a script will be interpreted by the operating system for an event handler with an action type of SCRIPT, if no script type is specified in the event handler definition. **Maximum acceptable return code** Lists the types of events supported by UEM Server. Default configuration options that are specific to the selected event type may be updated by selecting the Edit... button.
Default Component Name	Specifies the event driven UEM Server component (that is, those components with a component type of "uems") to which an event will be assigned if no component name is specified when the event definition is created. When this event driven UEM Server component starts, it will load and monitor those events that have been assigned to it. An event may not be assigned to a demand-driven UEM Server component (that is, those components with a component type of "uemd"), and only event-driven UEM Server components are listed here.
Tracking interval	Specifies the frequency with which a tracked event occurrence will be tested for completeness, if no such value is provided in the event definition. A value of 0 indicates that an event occurrence will not be tracked, and will be tested for completeness as soon as the occurrence is detected by UEM Server.
Set event-speci	Lists the types of events supported by UEM Server. Default configuration options that are specific to the selected event type may be updated by selecting the Edit... button.

fic optio ns	
Edit Button	Displays a dialog from which configuration options that are specific to the selected event type may be updated.  **Rename the file for a triggered event occurrence** Specifies the format that UEM Server should use when renaming a file for a triggered event occurrence, provided no such value was specified in the event definition. The value entered here may consist of literal values or one or more the variables listed below which UEM Server with applicable run-time values. • $(compname) the name of the UEM Server component • $(compid) the ID of the UEM Server component instance • $(date) the current date, in yyyymmdd format • $(time) the current time, in hhmmss format • $(origname) the original file name • $(origext) the file's original extension • $(seqnum) a value that begins at 1 and is incremented by 1 as each file is renamed by a given instance of UEM Server. **Minimum file size** Specifies the default file size a tracked file must be before it can be considered complete. That is, if a tracked file is tested for completeness and its file size is the same on two consecutive checks, the file will not be considered complete until it grows to at least the size specified here. This value may be overridden in the event definition.

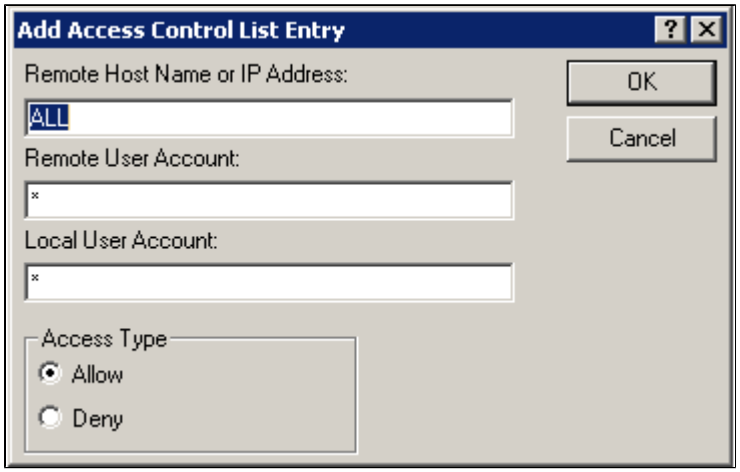
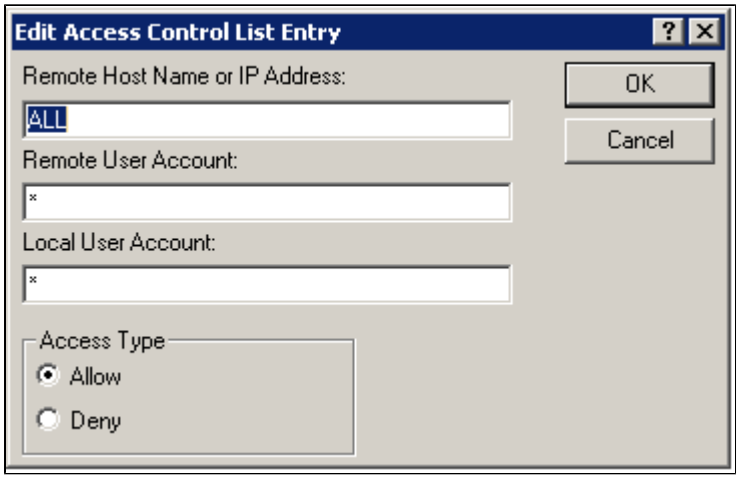
Universal Event Monitor Server: Access Control List



Universal Event Monitor Server: Access Control List: Access ACL

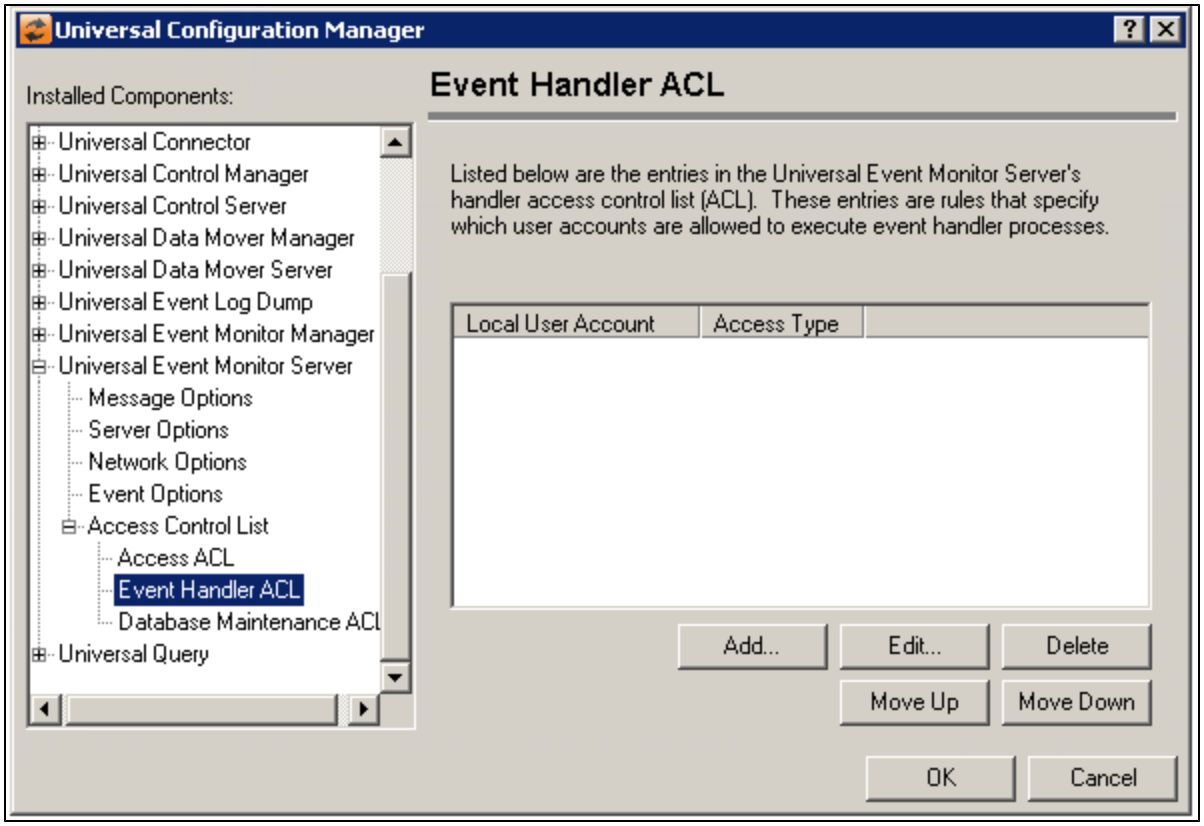


Field	Description
Access	Displays the entries contained in the Universal Event Monitor Server Access Control List (ACL). This list is used to grant or deny access to Universal Event Monitor Server based on the remote system's IP address, the user account running the Universal Event

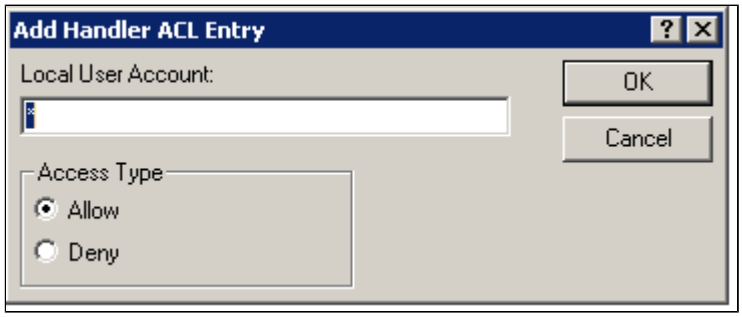
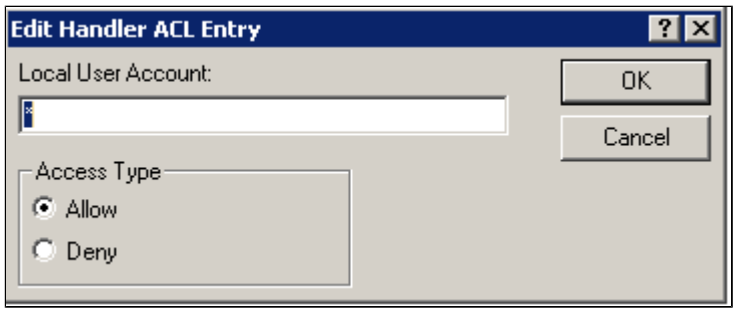
Control List (ACL)	Monitor Manager or Load Utility, and the user account with which the Server is executed. The order in which ACL entries are evaluated and applied are based on their position in this list. Use the Move Up and Move Down buttons to change an entry's priority.
Add Button	Displays a dialog box that allows an entry to be added to the Access Control List. The fields in this dialog are populated with default values.  **Remote Host Name / IP Address** The IP address or name of the remote host for which access will be allowed or denied. To apply an access control entry to all IP addresses, use the keyword ALL (in all caps). **Remote User Account** The ID of the user account executing the client application, either the Universal Event Monitor Manager or the Universal Event Monitor Load Utility. **Local User Account** The local user account that will be used to run the requested process. **Access Type** Will cause the Universal Control Server to accept/Reject incoming connections from the specified host, provided the Universal Control Manager is executed by the specified user account and the process is owned by the specified local user account.
Edit Button	Displays a dialog that allows the selected Access Control List entry to be modified.  **Remote Host Name / IP Address** The IP address or name of the remote host for which access will be allowed or denied. To apply an access control entry to all IP addresses, use the keyword ALL (in all caps). **Remote User Account** The ID of the user account executing the client application, either the Universal Event Monitor Manager or the Universal Event Monitor Load Utility.

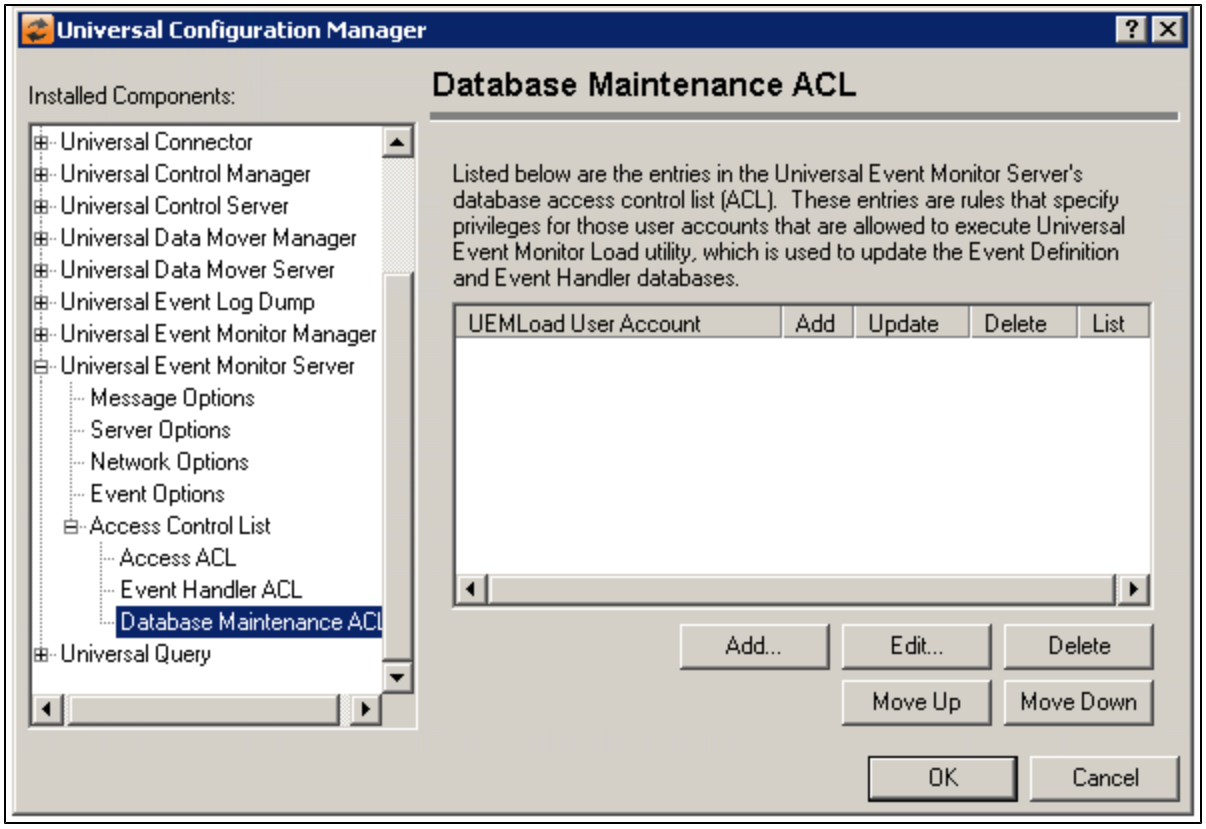
	Local User Account The local user account that will be used to run the requested process. **Access Type** Will cause the Universal Control Server to accept/Reject incoming connections from the specified host, provided the Universal Control Manager is executed by the specified user account and the process is owned by the specified local user account.
Delete Button	Removes the selected item from the Access Control List.
Move Up Button	Moves the selected item up one position in the list.
Move Down Button	Moves the selected item down one position in the list.

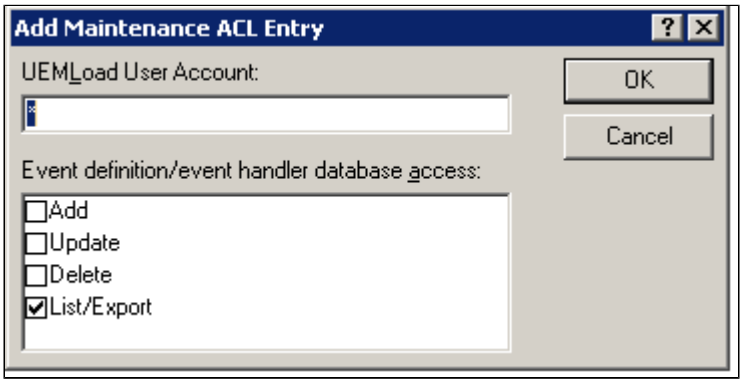
Universal Event Monitor Server: Access Control List - Event Handler ACL

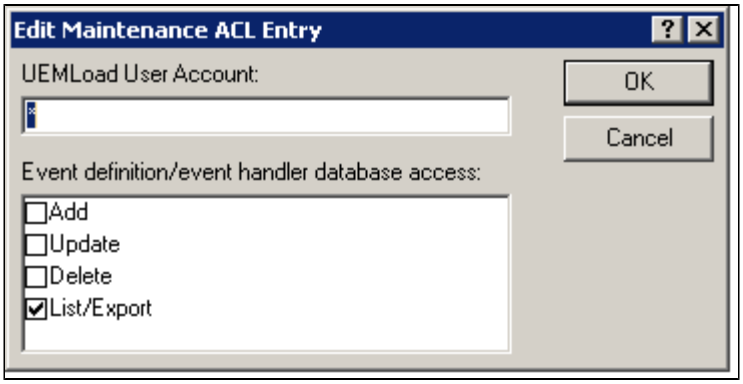


Field	Description
Event Handler Access Control List	Displays the entries contained in the Universal Event Monitor Server Handler Access Control List (ACL). This list is used to grant or deny access to execution of an Event Handler process based on the user account requested. The order in which ACL entries are evaluated and applied are based on their position in this list. Use the Move Up and Move Down buttons to change an entry's priority.
Add Button	Displays a dialog box that allows an entry to be added to the Access Control List. The fields in this dialog are populated with default values.

	 **Local User Account** The local user account that will be used to run the requested process. **Access Type** Will cause the Universal Event Monitor Server to allow/deny execution of a handler process for the specified user account.
Edit Button	Displays a dialog that allows the selected Access Control List entry to be modified.  **Local User Account** The local user account that will be used to run the requested process. **Access Type** Will cause the Universal Event Monitor Server to allow/deny execution of a handler process for the specified user account.
Delete Button	Removes the selected item from the Access Control List.
Move Up Button	Moves the selected item up one position in the list.
Move Down button	Moves the selected item down one position in the list.



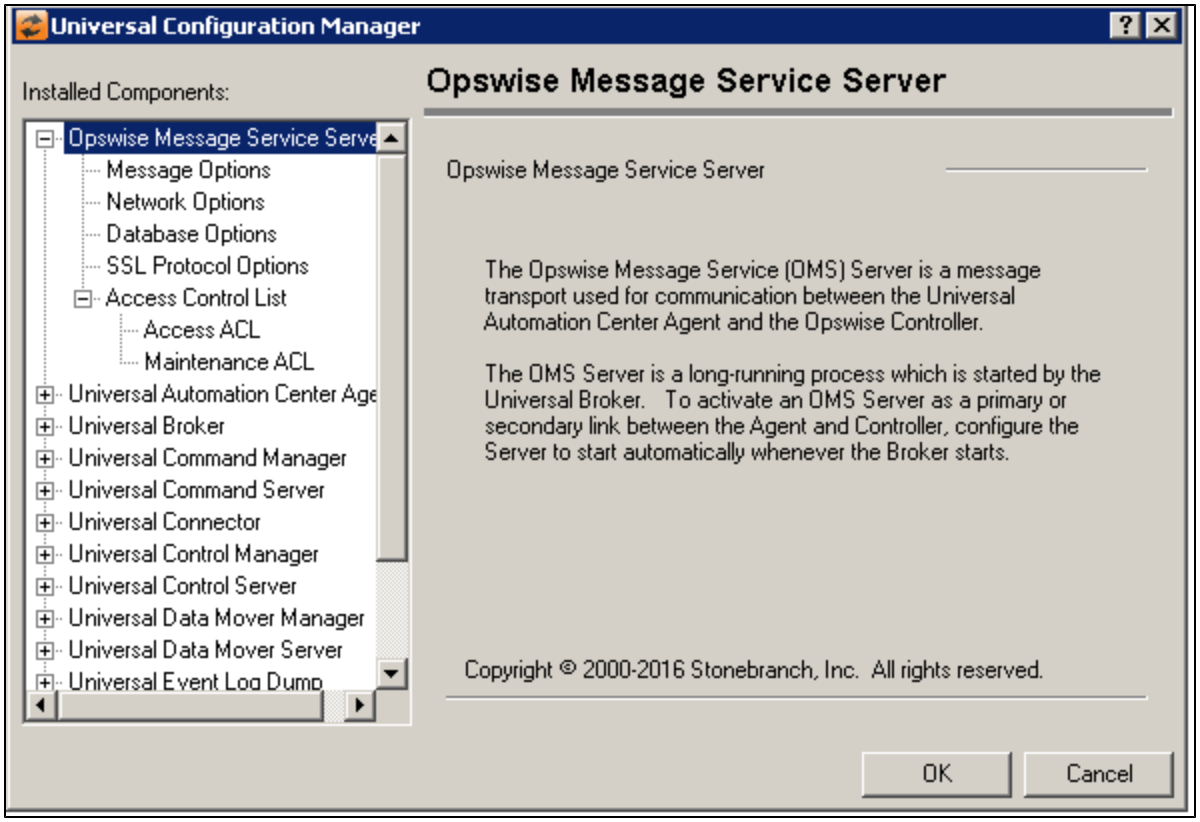
Field	Description
Database Maintenance Access Control List	Displays the entries contained in the Universal Event Monitor Server Maintenance Access Control List (ACL). This list is used to grant or deny access to Event Definition and Event Handler database updates based on the user account with which the UEMLoad utility is run. The order in which ACL entries are evaluated and applied are based on their position in this list. Use the Move Up and Move Down buttons to change an entry's priority.
Add Button	Displays a dialog box that allows an entry to be added to the Access Control List. The fields in this dialog are populated with default values.  **UEMLoad User Account** The account with which the Universal Event Monitor Load utility is run. This is the account for which database maintenance access is either allowed or denied. **Event definition/event handler database access** Specifies which activities the local user account may perform on the Event Definition and Event Handler databases via the Universal Event Monitor Load utility. Check the box next to the listed privilege to grant the user account that type of access. If the box next to the privilege type is not checked, the user account will be denied that access.
Edit Button	Displays a dialog that allows the selected Access Control List entry to be modified.

	 **UEMLoad User Account** The account with which the Universal Event Monitor Load utility is run. This is the account for which database maintenance access is either allowed or denied. **Event definition/event handler database access** Specifies which activities the local user account may perform on the Event Definition and Event Handler databases via the Universal Event Monitor Load utility. Check the box next to the listed privilege to grant the user account that type of access. If the box next to the privilege type is not checked, the user account will be denied that access.
Delete Button	Removes the selected item from the Access Control List.
Move Up Button	Moves the selected item up one position in the list.
Move Down button	Moves the selected item down one position in the list.

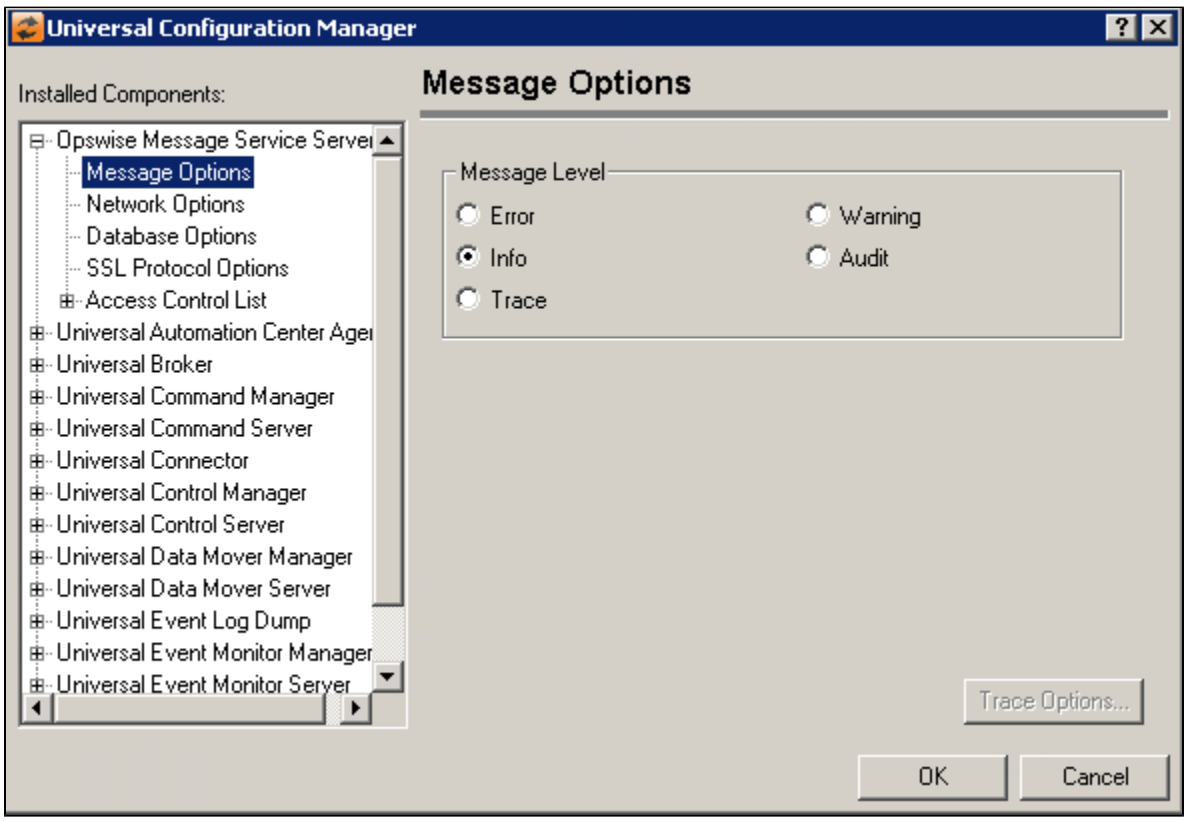
Universal Configuration Manager - Universal Message Service (OMS)

- Universal Message Service Server
- Universal Message Service Server: Message Options
- Universal Message Service Server: Network Options
- Universal Message Service Server: Database Options
- Universal Message Service Server: SSL/TLS Protocol Options
- Universal Message Service Server: Access Control List
  - Universal Message Service Server: Access Control List - Access ACL
  - Universal Message Service Server: Access Control List - Maintenance ACL

Universal Message Service Server



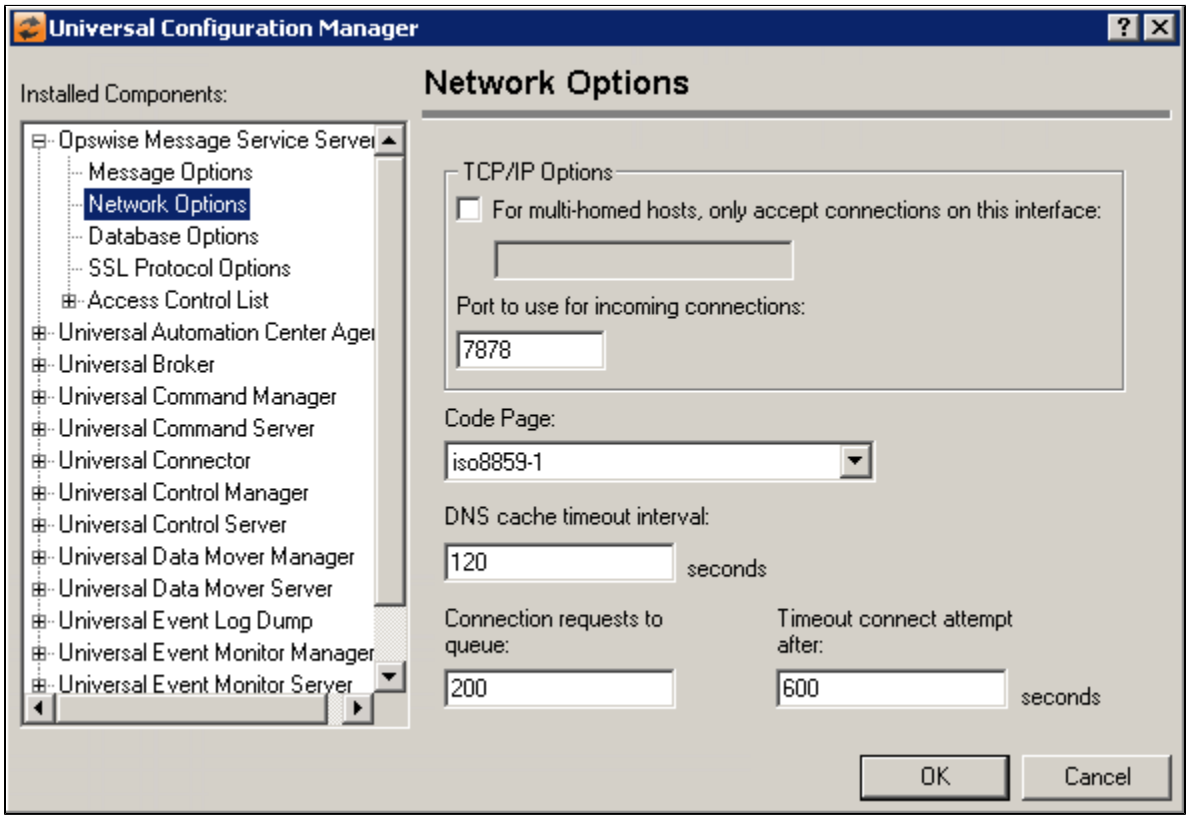
Universal Message Service Server: Message Options



Field	Description
Message Level	

	Error Turns on message logging for errors only. **Warning** Turns on message logging for warnings and errors. **Info** Turns on message logging for all informational, warning and error messages. **Audit** Turns on message logging for all audit, informational, warning and error messages. **Trace** Allows all messages to be logged, including those used for debugging purposes.
Trace Options Button	Displays a dialog that allows options for trace statement buffering and trace file limits to be set.

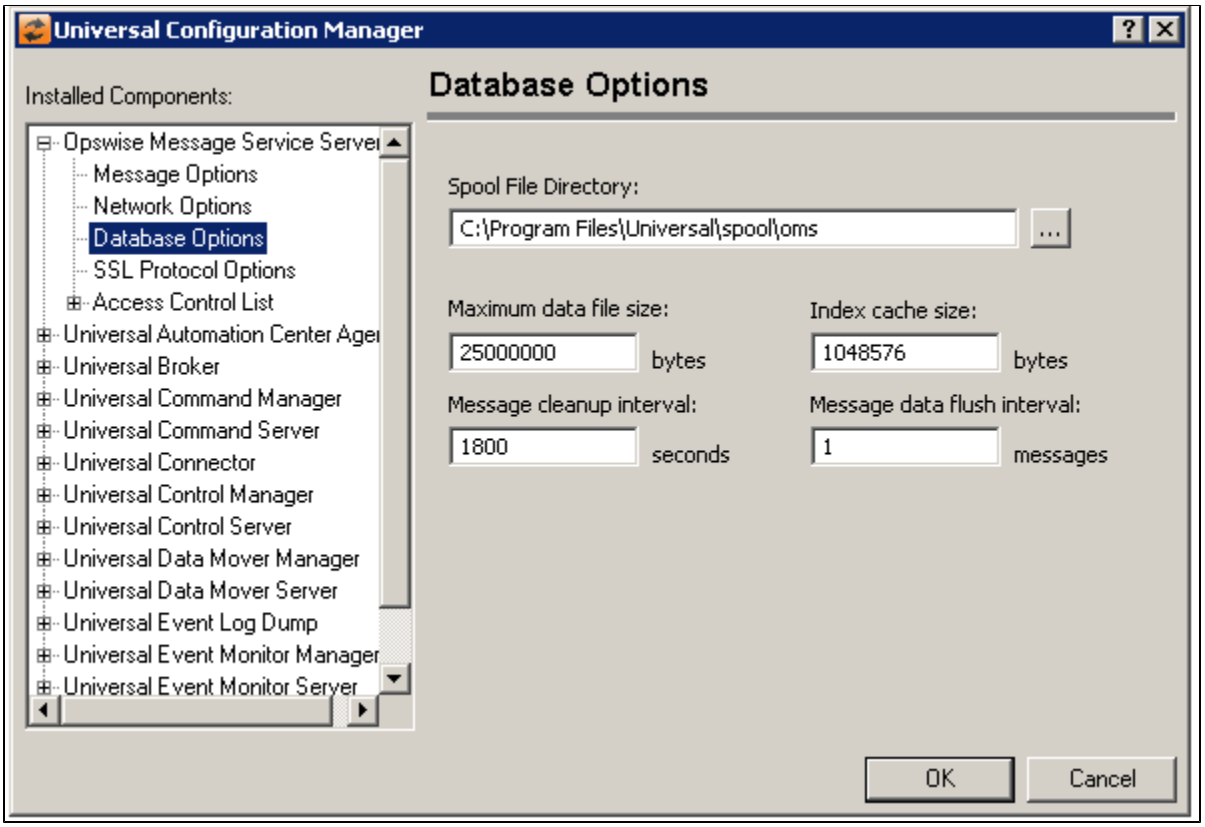
Universal Message Service Server: Network Options



Field	Description
TCP/IP Options	**multi-homed host** A multi-homed host is a machine that has more than one network interface card installed. On such machines, OMS Server may be configured to accept incoming connections on only one of the interfaces. To specify this address, check the option and enter a valid host name or IP address in the space provided. Otherwise, leave the option unchecked, and OMS Server will accept connections on all network interfaces. **IP Address** Contains the IP address or host name of the interface on which OMS Server is accepting connections. **Port** Enter a port number or service name that will be used to accept incoming connections from OMS clients and/or peers.
Code Page	Sets the translation table used to translate text-based data transmitted across the network.
DNS Cache	Specifies the number of seconds to retain a resolved host name in the OMS Server memory cache. A value of 0 specifies no caching.

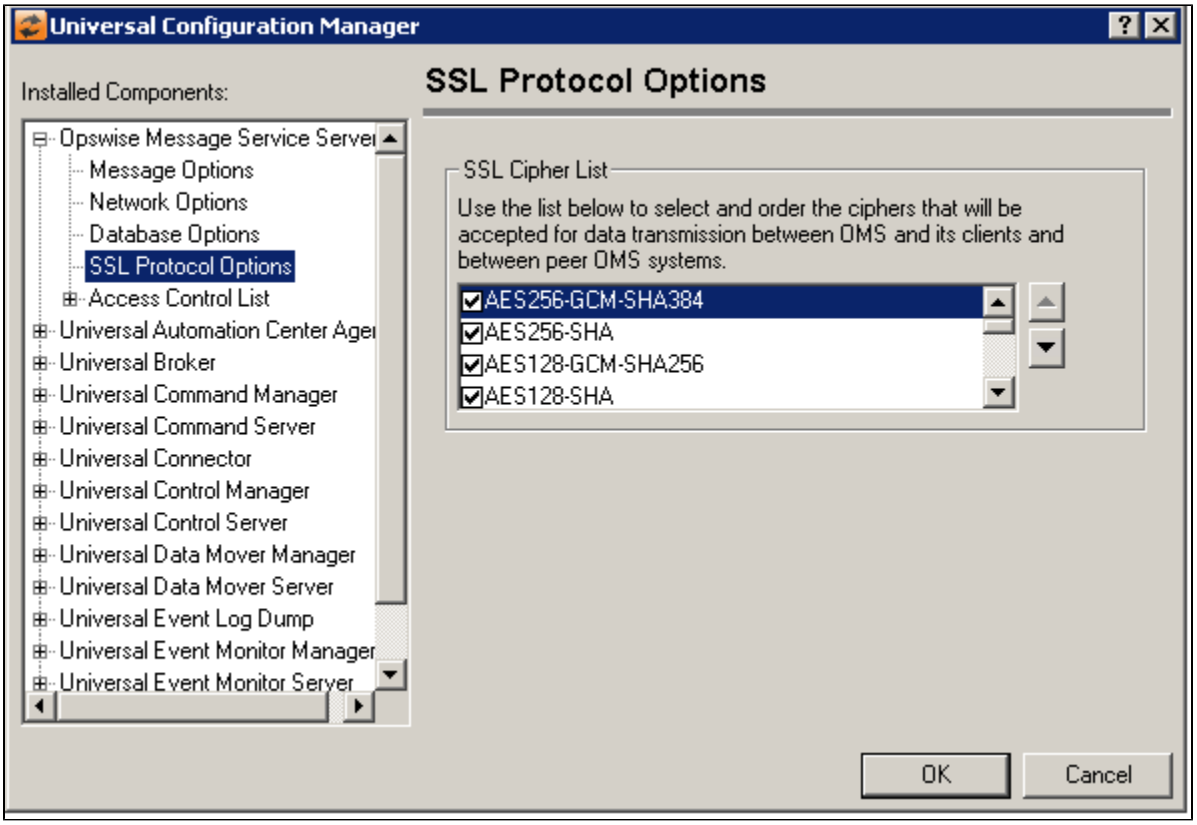
Timeout interval	
Connection Requests to queue	Specifies the number of connection requests that will be queued by OMS Server. When the number of pending connection requests reaches this value, additional connection requests will be denied and a "connection refused" error will be issued.
Timeout connection attempts after	The service interface timeout value specifies the number of seconds of inactivity before the OMS server considers the connection as inactive and closes the connection. A connection is considered inactive if no data is sent or received over the socket connection.

Universal Message Service Server: Database Options



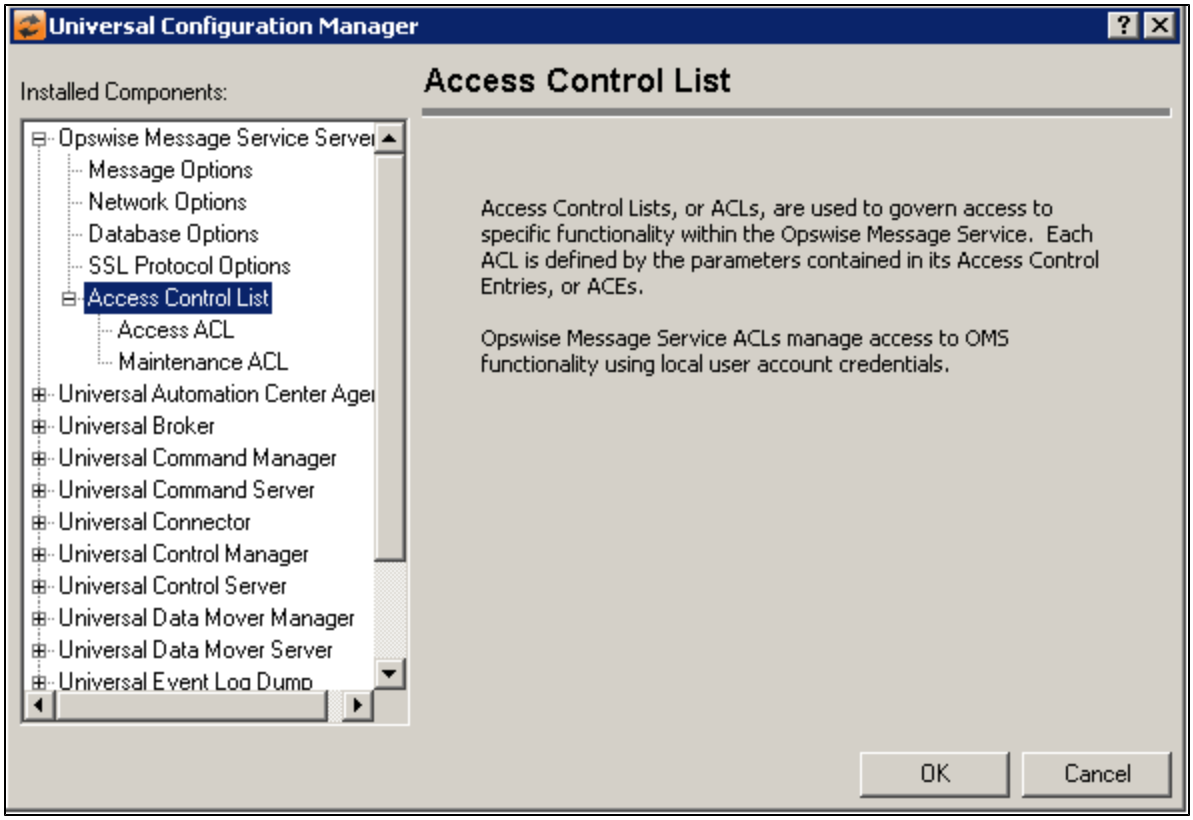
Field	Description
Spool File Directory	The directory where the spool files are stored. This directory must be located on local storage. It may not be located on any storage area accessed via the network which includes but is not limited to NFS, SAMBA, Microsoft File Sharing.
Maximum data file size	The maximum size, in bytes, of the message data file. Once the maximum size is reached, OMS Server creates a new data file.
Index cache size	The message index provides quick access to the messages in the message data files. Additionally, the message index stores non-persistent messages. The message index cache size specifies the size of the memory cache used to store the index.
Message Cleanup interval	Message data file cleanup consists of examining each message data file and removing empty message data files. Cleanup is performed periodically, using the number of seconds specified here.
Message Data Flush Interval	(No help available.)

Universal Message Service Server: SSL/TLS Protocol Options

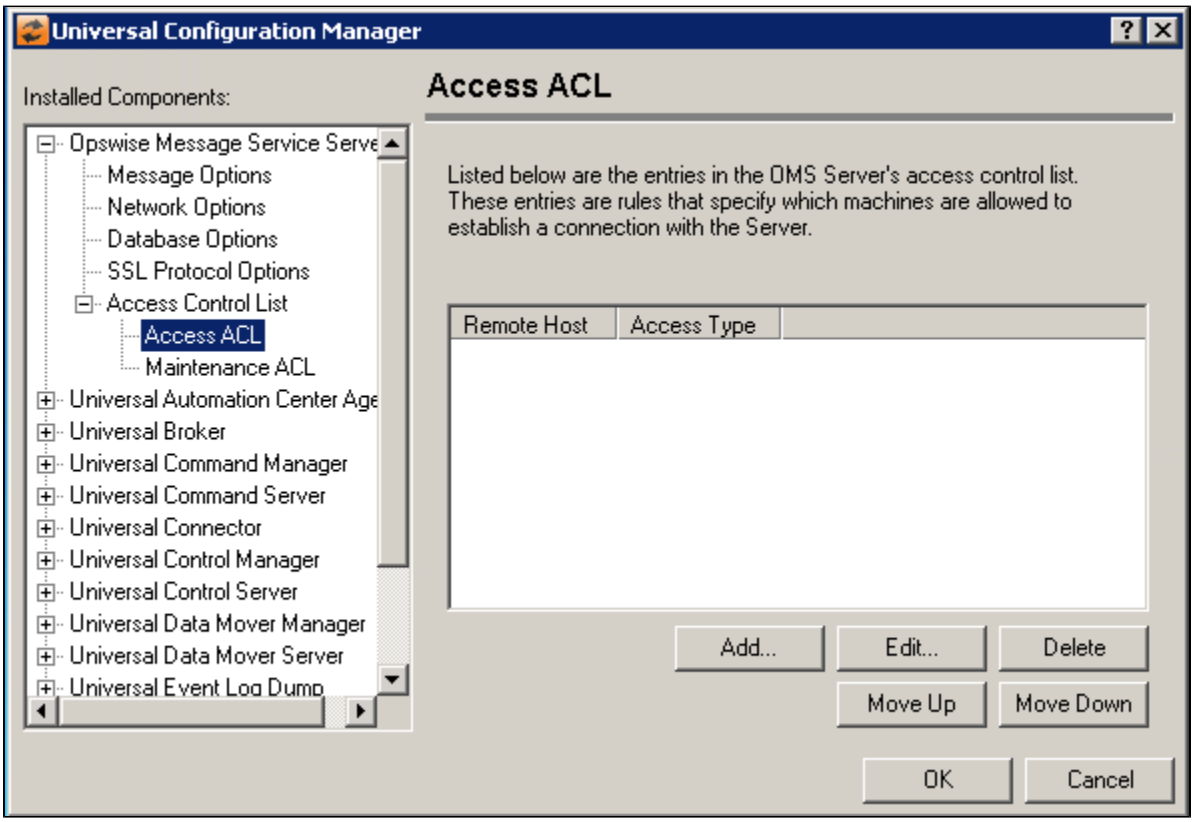


Field	Description
SSL Cipher List	The SSL/TLS cipher list specifies one or more acceptable cipher suites to use for network communication. The list consists of one or more cipher suites ordered with the most preferred suite first and the least preferred suite last.

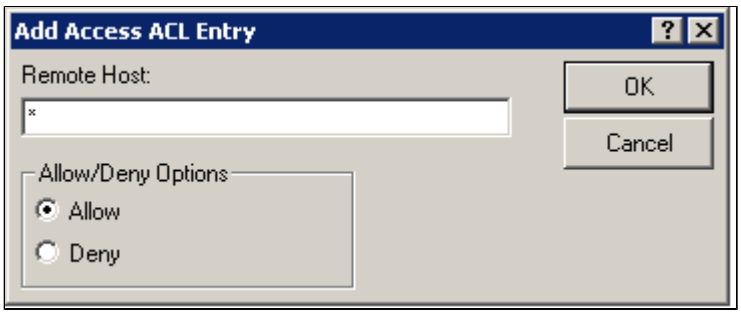
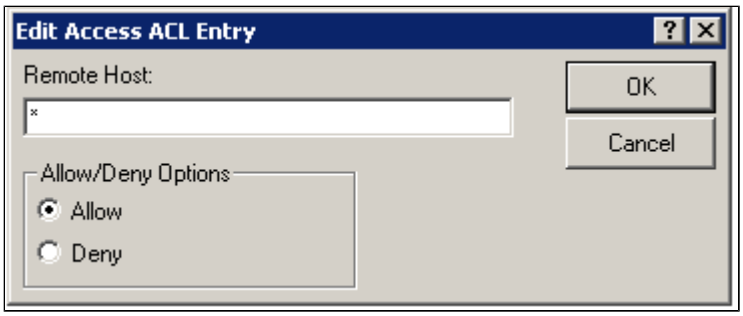
Universal Message Service Server: Access Control List

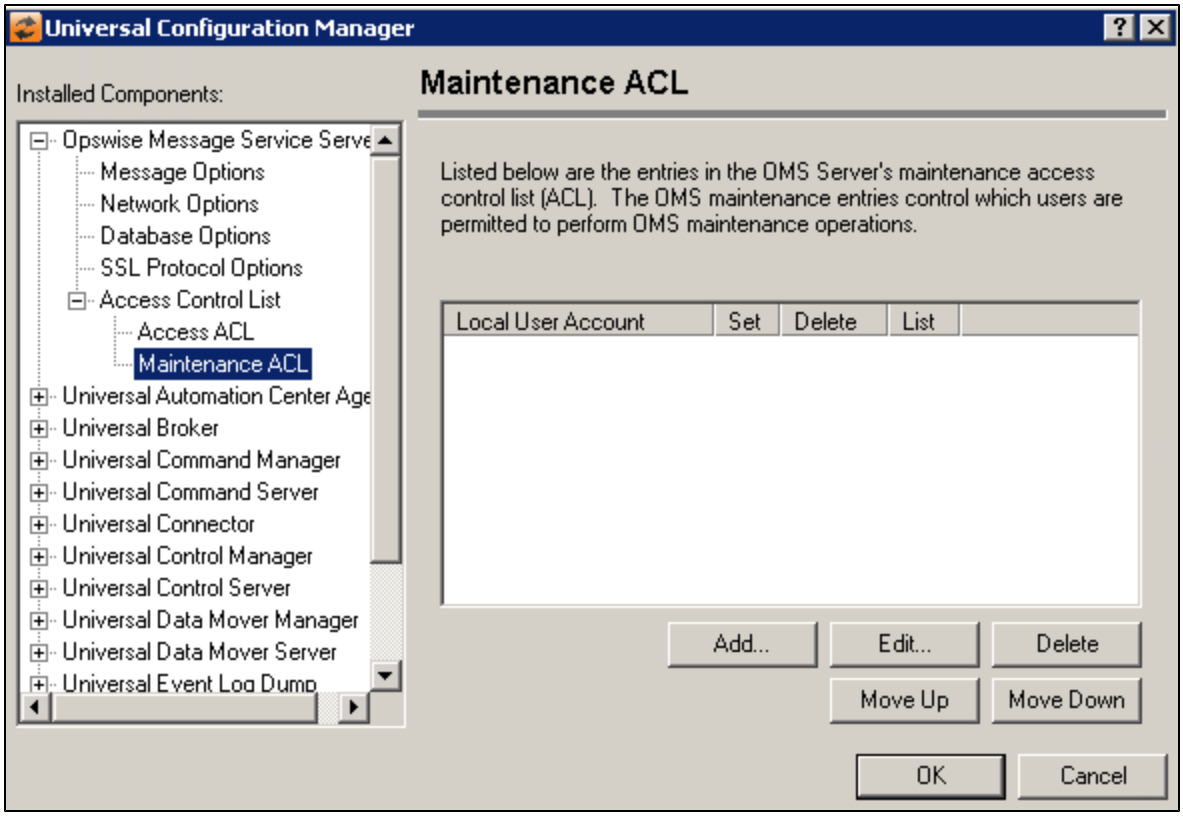


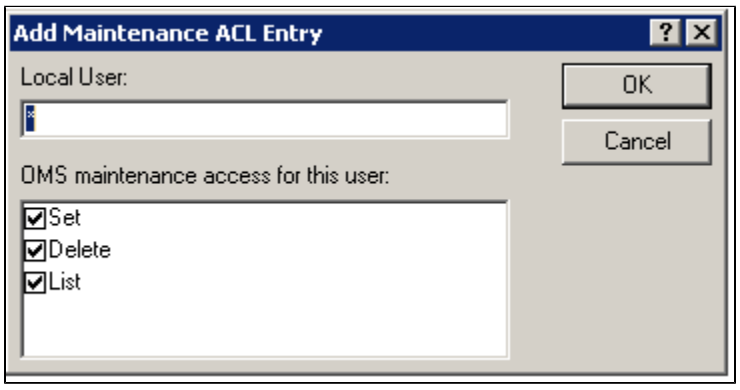
Universal Message Service Server: Access Control List - Access ACL

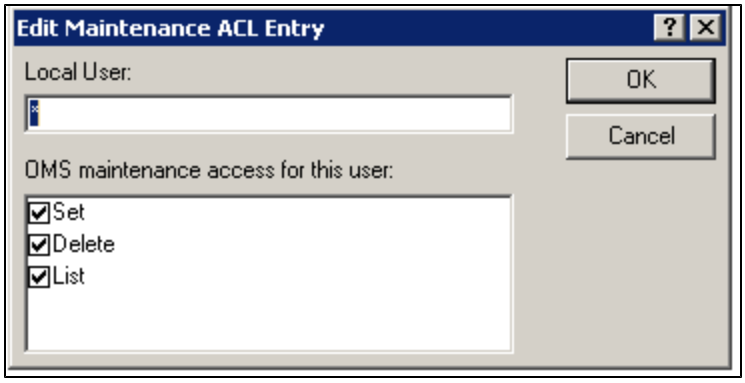


Field	Description
OMS Serv	

er ACL List	Displays the entries contained in the Universal Message Service (OMS) Server Access Control List (ACL). This list is used to grant or deny access to the local OMS Server based on the remote system's host name IP address. The order in which ACL entries are evaluated and applied are based on their position in this list. Use the Move Up and Move Down buttons to change an entry's priority.
Add Button	Displays a dialog box that allows an entry to be added to the Access Control List. The fields in this dialog are populated with default values.  **Remote Host** The host name or IP address of the remote system upon which a Universal Agent resides. The access specified for that agent determines whether it has the permission to connect to the local OMS Server. **Allow/Deny Options** Grants/prevents the Universal Agent on the specified remote host the permission to connect to the local OMS Server.
Edit button	Displays a dialog that allows the selected Access Control List entry to be modified.  **Remote Host** The host name or IP address of the remote system upon which a Universal Agent resides. The access specified for that agent determines whether it has the permission to connect to the local OMS Server. **Allow/Deny Options** Grants/prevents the Universal Agent on the specified remote host the permission to connect to the local OMS Server.
Delete Button	Removes the selected item from the Access Control List.
Move Up Button	Moves the selected item up one position in the list.
Move Down Button	Moves the selected item down one position in the list.



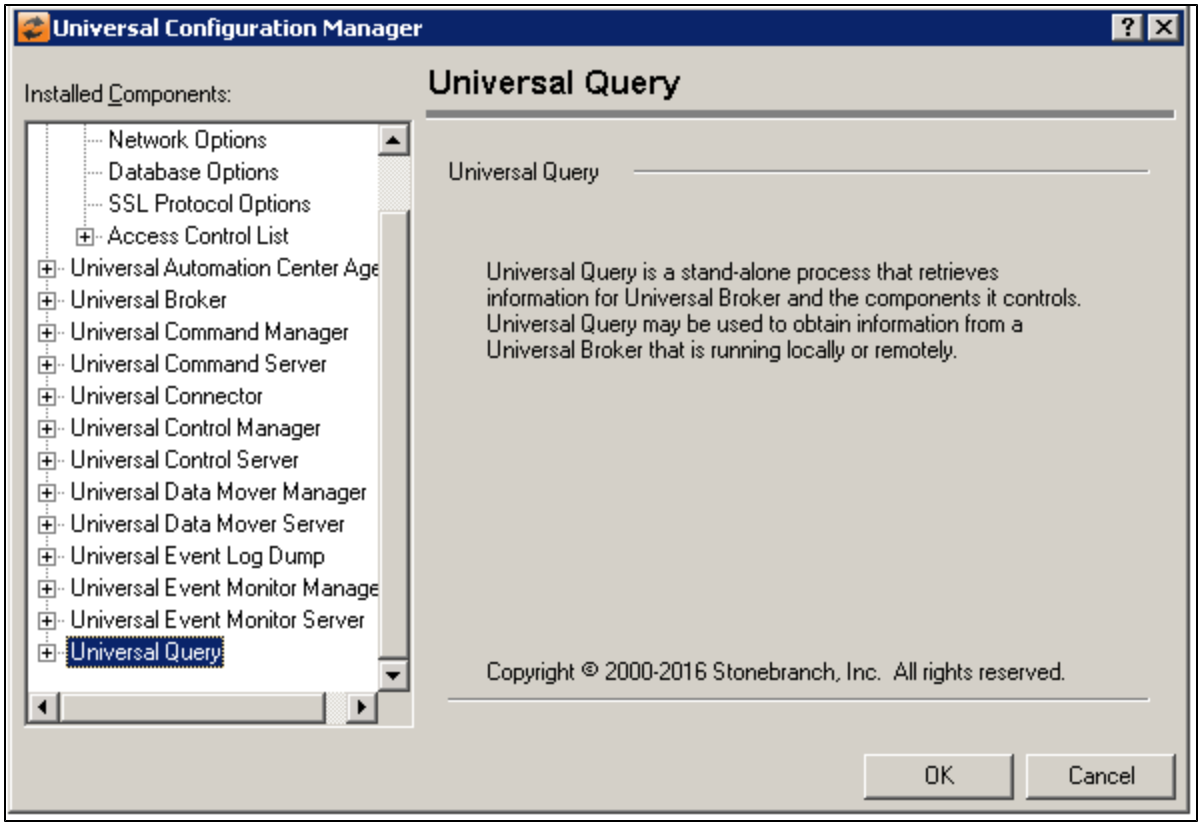
Field	Description
Maintenance ACL List	Displays the entries contained in the Universal Message Service (OMS) Server Maintenance Access Control List (ACL). This list is used to grant or deny access to OMS Server maintenance based upon the user account with which the maintenance is executed. The order in which ACL entries are evaluated and applied are based on their position in this list. Use the Move Up and Move Down buttons to change an entry's priority.
Add Button	Displays a dialog box that allows an entry to be added to the Access Control List. The fields in this dialog are populated with default values.  **Local User** The local account with which the OMS maintenance is performed. This is the account for which maintenance access is either allowed or denied. **OMS maintenance access for this user** Specifies which activities the local user account may perform against the local OMS Server. Check the box next to the listed privilege to grant the user account that type of access. If the box next to the privilege type is not checked, the user account will be denied that access.
Edit button	Displays a dialog that allows the selected Access Control List entry to be modified.

	 **Local User** The local account with which the OMS maintenance is performed. This is the account for which maintenance access is either allowed or denied. **OMS maintenance access for this user** Specifies which activities the local user account may perform against the local OMS Server. Check the box next to the listed privilege to grant the user account that type of access. If the box next to the privilege type is not checked, the user account will be denied that access.
Delete Button	Removes the selected item from the Access Control List.
Move Up Button	Moves the selected item up one position in the list.
Move Down Button	Moves the selected item down one position in the list.

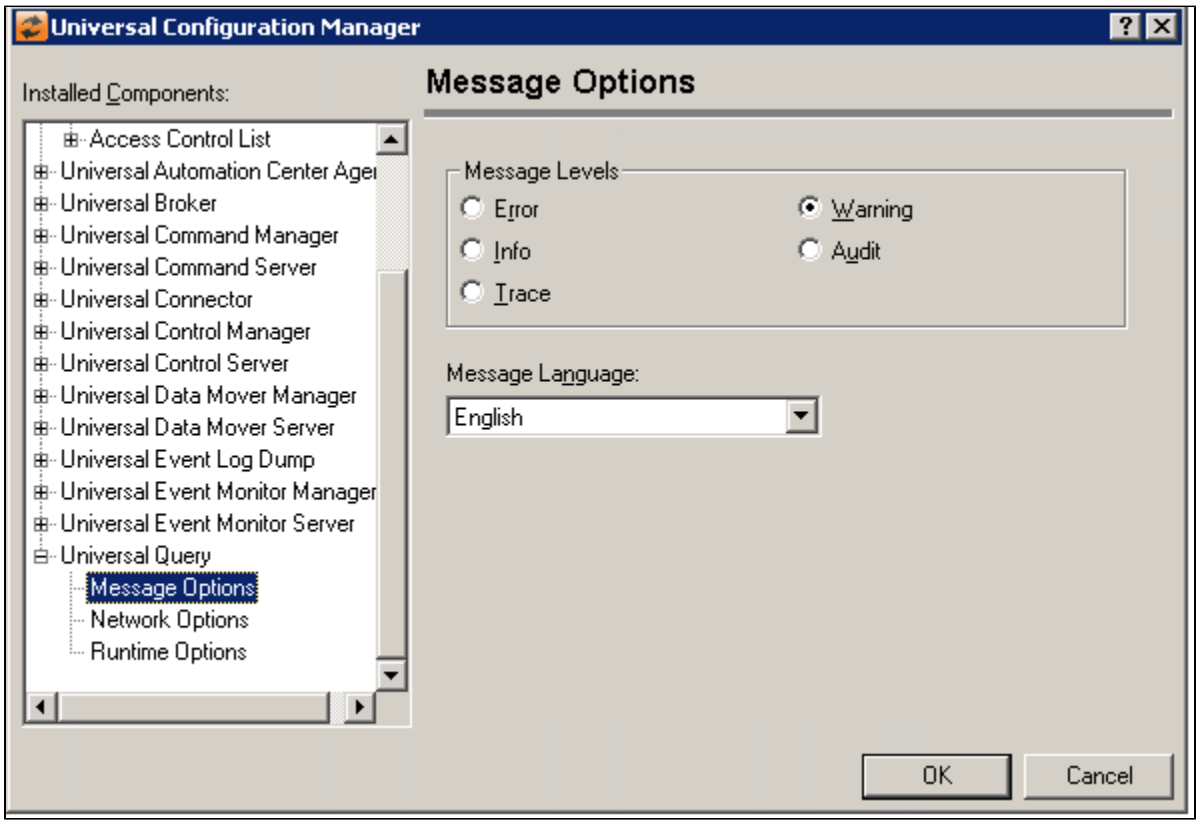
Universal Configuration Manager - Universal Query

- Universal Query
- Universal Query: Message Options
- Universal Query: Network Options
- Universal Query: Runtime Options

Universal Query



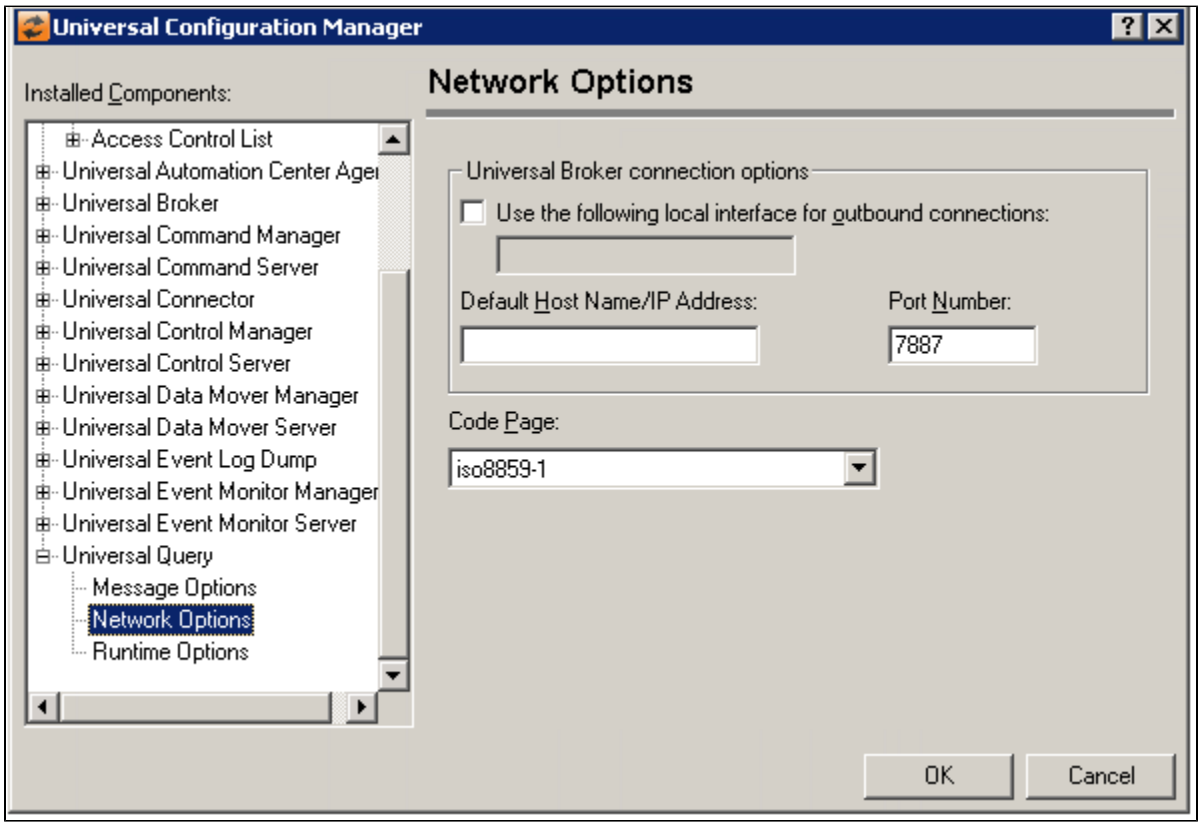
Universal Query: Message Options



Field	Description
Message Level	

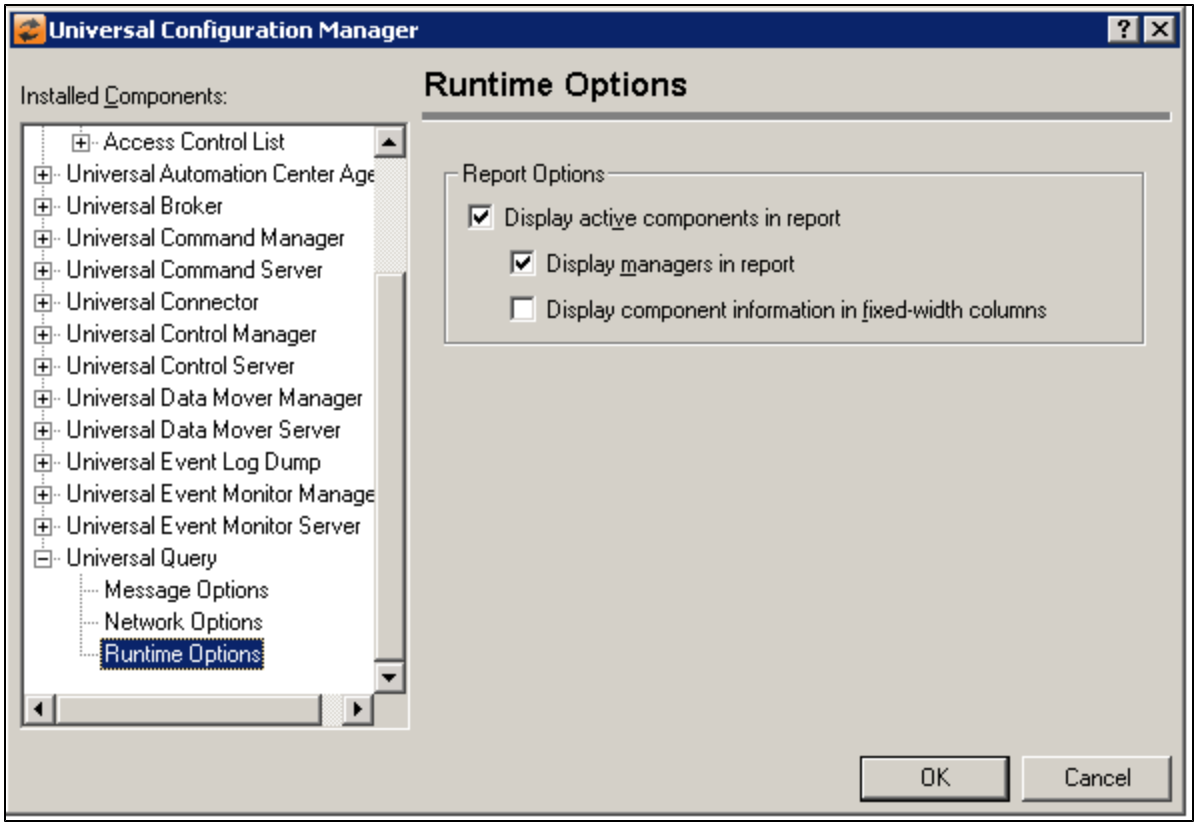
	Error Turns on message logging for errors only. **Warning** Turns on message logging for warnings and errors. **Info** Turns on message logging for all informational, warning and error messages. **Audit** Turns on message logging for all audit, informational, warning and error messages. **Trace** Allows all messages to be logged, including those used for debugging purposes.
Message Language	Specifies the language used to display error messages.

Universal Query: Network Options



Field	Description
Universal Broker Connection Options	**Outbound IP Option** When this option is selected, the IP address or alias of a local interface may be specified that will be used whenever Universal Query connects to a Universal Broker.If this option is not selected, any available local interface will be used. **Default Host Name/IP Address** Enter a valid IP address or host name that will be used as the default system for process execution. To run processes on the local system, enter a host name of localhost, or an IP address of 127.0.0.1. **Port Number** Enter a port number or service name that will be used to connect to Universal Broker.
Code Page	Sets the translation table used to translate text-based data transmitted across the network.

Universal Query: Runtime Options



Field	Description
Report options	**Display Active Components in report** Check this option if you want to display information for the active components currently being managed by a specific Universal Broker. If this option is not checked, only the status of the Broker will be displayed. **Display managers in report** heck this option if you want to display information for Universal Manager components and utilities currently registered with a specified Universal Broker. If this option is not checked, only the status of the Universal Server components will be displayed. **Display component information in fixed-width columns** Check this option to display information for active components in fixed-width colums. This will allow the information in the resulting text file to be easily parsed. If this option is not checked, active component information will be formatted in variable-width columns, as determined by the information returned.

Configuration Refresh

- Overview
- Configuration Refresh via Universal Control
  - Configuration Refresh via Universal Control for Universal Event Monitor Server
  - Configuration Refresh via Universal Control for Universal Automation Center Agent
- Configuration Refresh via I-Management Console
- Configuration Refresh via Universal Configuration Manager
- Configuration Refresh of Universal Broker for its own Configuration Options

Overview

Universal Broker maintains the configuration files for all Universal Agent components, including itself. The components do not read their configuration files themselves (except for Universal Enterprise Controller, which does read its own configuration file).

At initial start-up, Universal Broker reads the configuration files of all components and places the configuration data in Universal Broker memory. When a component starts, it first registers with its local Universal Broker. As part of the registration process, Universal Broker sends that component's configuration data to the component. Thereafter, if a configuration file is modified, Universal Broker must be refreshed. This directs Universal Broker to re-read all component configuration files and update the configuration data in memory.

A Universal Broker is refreshed when any of the following occurs:

- Universal Broker is recycled (stopped and restarted).
- Universal Broker is refreshed via Universal Control.
- Universal Broker is refreshed via the I-Management Console UEC client application.
- Universal Broker is refreshed via Universal Configuration Manager (**Windows only**).

Then, when a component restarts, it again registers with its local Universal Broker, which sends that component's configuration data to the component.

Configuration Refresh via Universal Control

Universal Control refreshes the Universal Broker by issuing a configuration refresh request via its REFRESH_CMD configuration option.

The configuration refresh request directs Universal Broker to take the following actions:

Step 1	Read all component configuration files, or the configuration file of a single, specified component, and update the configuration data in Universal Broker memory. (Currently, the only individual component that can be refreshed this way is the Universal Event Monitor Server, **uems**.)
Step 2	Read all component definitions (or only the component definition for **uems**) in the Component Definition directory. Universal Broker replaces all component definitions with the newly read component definitions. New component definitions are added and deleted component definitions are removed.
Step 3	Read the Universal Access Control List (UACL) configuration entries (or only entries for **uems**) from the registry. Universal Broker replaces its UACL entries with the newly read entries.

Configuration Refresh via Universal Control for Universal Event Monitor Server

Because an event-driven Universal Event Monitor Server typically is a long-running process, the ability to refresh an active UEM Server's configuration and list of assigned event definitions is provided. Automatic refresh of configuration and event information for a demand-driven UEM Server is not supported; the values it obtains at start-up are the ones it uses throughout its lifetime.

When a change is made to the stored UEM Server configuration settings (see Configuration File), active event-driven UEM Servers must be notified that a change has taken place. This is done via Universal Control, using the Universal Control Manager REFRESH_CMD option, along with a component type value that identifies the component to refresh (see Refreshing via Universal Control Examples).

Windows
A request to update the configuration of local event-driven UEM Servers is issued automatically whenever a change is made to a UEM Server's configuration through the Universal Configuration Manager.

When Universal Control or the Universal Configuration Manager (Windows only) instructs an active event-driven UEM Server to refresh its cached configuration, the event-driven Server processes the request immediately.

The UEMLoad utility automatically notifies an event-driven UEM Server of an event definition change via a flag that resides in the local Universal Broker. UEM Server checks this flag every two minutes and updates its cached list of event definitions whenever UEMLoad updates them. This eliminates the need to refresh UEM Server with Universal Control following a database change.

Configuration Refresh via Universal Control for Universal Automation Center Agent

Since Universal Automation Center Agent (UAG) starts automatically when the Universal Broker starts, the ability to refresh an active Universal Automation Center Agent's configuration is provided.

When a change is made to the stored UAG configuration settings (see Configuration File), active UAG components must be notified that a change has taken place. This is done via Universal Control, using the Universal Control Manager REFRESH_CMD option, along with a component type value that identifies the component to refresh (see Refreshing via Universal Control Examples).

Windows
A request to update the configuration of UAG is issued automatically whenever a change is made to UAG configuration through the Universal Configuration Manager.

When Universal Control or the Universal Configuration Manager (Windows only) instructs UAG to refresh its cached configuration, UAG processes the request immediately.

The following UAG configuration file options are dynamically updated by Universal Control Manager REFRESH_CMD:

Option	Description
LOGLVL	Logging level of UAG.
MESSAGE_LEVEL	Level of messages to write for UAG.
SECURITY	Activates user security.
PROCESS_CANCEL_TIMEOUT	Timeout period between CANCEL request and force termination.
TRACE_DIRECTORY	Directory that UAG uses for trace files (MESSAGE_LEVEL option value is set to trace).
TRACE_FILE_LINES	Maximum number of lines to write to the trace file.
TRACE_TABLE	Size of a wrap-around trace table maintained in memory.

Additionally, the following UAG UACL file entry is dynamically updated by Universal Control Manager REFRESH_CMD:

Entry	Description
UAG_WORK_REQUEST	Allows or denies access to a task execution request and if allowed, specifies whether or not user authentication is performed.

Configuration Refresh via I-Management Console

When configuration options are updated using the I-Management Console, a configuration refresh request automatically is sent to Universal Broker, and its configuration data is refreshed.

The configuration refresh request directs Universal Broker to take the following actions:

Step 1	Read all component configuration files and update the configuration data in Universal Broker memory.
Step 2	Read all component definitions in the Component Definition directory. Universal Broker replaces all component definitions with the newly read component definitions. New component definitions are added and deleted component definitions are removed.
Step 3	Read the Universal Access Control List (UACL) configuration entries from the registry. Universal Broker replaces its UACL entries with the newly read entries.

Configuration Refresh via Universal Configuration Manager

When configuration options are updated using the Universal Configuration Manager, a configuration refresh request is sent to Universal Broker, and its configuration is refreshed automatically.

The configuration refresh request directs Universal Broker to take the following actions:

Step 1	Read its configuration file. Universal Broker refreshes its configuration options.
Step 2	Read all component definitions in the Component Definition directory. Universal Broker replaces all component definitions with the newly read component definitions. New component definitions are added and deleted component definitions are removed.
Step 3	Read the Universal Access Control List (UACL) configuration entries from the registry. The Broker replaces its UACL entries with the newly read entries.

Configuration Refresh of Universal Broker for its own Configuration Options

As with all Universal Agent components, all Universal Broker options can be modified by editing its configuration file (**ubroker.conf**) directly.

However, unlike other components, not all Universal Broker options can be modified via I-Management Console. (In I-Management Console, these Universal Broker options are read-only.). These options can be modified only by editing the Universal Broker configuration file, **`ubroker.conf`**. For these modifications to be updated in Universal Broker memory and take immediate effect, Universal Broker must be recycled.

All other Universal Broker options can be modified either:

- By editing **ubroker.conf**.
- Via I-Management Console.
- Via the Universal Configuration Manager.

Depending on the configuration option, for a modification to be updated in Universal Broker memory and take immediate effect:

- Universal Broker must be recycled.
- Universal Broker must be refreshed by issuing a Universal Control configuration refresh request (via the REFRESH_CMD configuration option), if the modifications are made in **ubroker.conf**.
- Universal Broker is refreshed automatically, if the modifications are made via I-Management Console or the Universal Configuration Manager.

For a list of the Universal Broker configuration options in each category, see Universal Broker Configuration Options Refresh.

Refreshing via Universal Control Examples

Refreshing via Universal Control Examples

- Refreshing Universal Broker from z/OS
- Refreshing a Component from z/OS
- Refreshing Universal Broker from Windows
- Refreshing a Component from Windows
- Refreshing Universal Broker from UNIX
- Refreshing a Component from UNIX
- Refreshing Universal Broker from IBM i
- Refreshing a Component from IBM i

These examples illustrate how to use Universal Control to refresh configuration data of all components, including itself, or a single component.

Currently, the only individual components that can be refreshed are the Universal Event Monitor Server (uems) and the Universal Automation Center Agent (uag).

Note
The IBM i examples reference the IBM i commands by their untagged names. If you are using commands with tagged names to run Universal Control, substitute the tagged names for these untagged names. (For information on tagged names, see UCHGRLS (Change Release Tag) Program.)

Refreshing Universal Broker from zOS

- Refreshing Universal Broker from z/OS
  - SYSIN Options
  - Universal Broker Actions
  - Components

Refreshing Universal Broker from z/OS

This example refreshes Universal Broker on z/OS.

```
//jobname  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*****************************************************************
//* (c) Copyright 2001-2008, Stonebranch, Inc.  All rights reserved.
//*
//* Stonebranch, Inc.
//* Universal Control
//*
//* Description
//* -----------
//* This sample demonstrates the use of the UCTL program to refresh
//* a running component on host dallas.
//*
//* Make the following modifications as required by your local
//* environment:
```

```
//*
//* - Modify the JOB statement as appropriate.
//* - Change all '#HLQ' to the high-level qualifier of the
//*   Universal Command data sets.
//* - If not already done, modify the JCL procedure UCTLPRC
//*   as required by your local environment.
//*************************************************************
//*
//         JCLLIB ORDER=#HLQ.UNV.SUNVSAMP
//*
//STEP1    EXEC UCTLPRC
//SYSIN    DD  *
 -refresh -host dallas
/*
```

This example refreshes the Universal Broker configuration on host **dallas**.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-refresh	Instruction to refresh Universal Broker on the remote system.
-host	Directs the command to a computer with a host name of **dallas**.

Universal Broker Actions

The refresh request directs the Broker to take the following actions:

Step 1	Read its configuration file. The Broker refreshes configuration options.
Step 2	Read all component definitions found in ddname **UNVCONF**. The Broker replaces all component definitions with the newly read component definitions. New component definitions are added and deleted component definitions are removed.
Step 3	Read the Universal Access Control List configuration file allocated to ddname **UNVACL**. The Broker replaces its UACL entries with the newly read entries.

Components

Universal Control

Refreshing a Component from zOS

- Refreshing a Component from z/OS
  - SYSIN Options
  - Components

Refreshing a Component from z/OS

This example refreshes a Universal Event Monitor Server (uems) component on a remote system.

```
//jobname  JOB CLASS=A,MSGCLASS=X
//STEP1    EXEC UCTLPRC
//SYSIN    DD  *
-refresh uems -cmdid "ABC-dallas" -host dallas -userid joe -pwd akkSdiq
/*
```

This example refreshes a Universal Automation Center Agent (uag) component on a remote system.

```
//jobname  JOB CLASS=A,MSGCLASS=X
//STEP1    EXEC UCTLPRC
//SYSIN    DD  *
-refresh uag -cmdid "ABC-dallas" -host dallas -userid joe -pwd akkSdiq
/*
```

SYSIN Options

The SYSIN options used in these examples are:

Option	Description
-refresh	Type of component to refresh on the remote system.
-cmdid	Assigns a command identifier of "ABC-dallas" to the started component.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the Universal Control Server process.
-pwd	Password for the user ID.

Components

Universal Control

Refreshing Universal Broker from Windows

- Refreshing Universal Broker via Universal Control from Windows
  - Command Line Options
  - Universal Broker Actions
  - Components

Refreshing Universal Broker via Universal Control from Windows

This example refreshes Universal Broker on a remote system.

```
uctl -refresh -host dallas -userid joe -pwd akkSdiq
```

Command Line Options

The command line options used in this example are:

Option	Description
-refresh	Instruction to refresh Universal Broker on the remote system.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the Universal Control Server process.
-pwd	Password for the user ID.

Universal Broker Actions

This refresh request directs Universal Broker to take the following actions:

Step 1	Read its configuration file. Universal Broker refreshes its configuration options.

Step 2	Read all component definitions found in the component definition directory. Universal Broker replaces all component definitions with the newly read component definitions. New component definitions are added and deleted component definitions are removed.
Step 3	Read the Universal Access Control List configuration entries from the registry. Universal Broker replaces its UACL entries with the newly read entries.

Components

Universal Control

Refreshing a Component from Windows

- Refreshing a Component via Universal Control from Windows
  - Command Line Options
  - Components

Refreshing a Component via Universal Control from Windows

This example refreshes a Universal Event Monitor Server (uems) component on a remote system.

```
uctl -refresh uems -cmdid "ABC-dallas" -host dallas -userid joe -pwd akkSdiq
```

This example refreshes a Universal Automation Center Agent (uag) component on a remote system.

```
uctl -refresh uag -cmdid "ABC-dallas" -host dallas -userid joe -pwd akkSdiq
```

Command Line Options

The command line options used in this example are:

Option	Description
-refresh	Type of component to refresh on the remote system.
-cmd	Assigns a command identifier of **"ABC-dallas"** to the started component.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the Universal Control Server process.
-pwd	Password for the user ID.

Components

Universal Control

Refreshing Universal Broker from UNIX

- Refreshing Universal Broker via Universal Control from UNIX
  - Command Line Options
  - Universal Broker Actions
  - Components

Refreshing Universal Broker via Universal Control from UNIX

This example refreshes Universal Broker on a remote system.

```
uctl -refresh -host dallas -userid joe -pwd akkSdiq
```

Command Line Options

The command line options used in this example are:

Option	Description
-refresh	Instruction to refresh Universal Broker on the remote system.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the Universal Control Server process.
-pwd	Password for the user ID.

Universal Broker Actions

This refresh request directs Universal Broker to take the following actions:

Step 1	Read its configuration file **ubroker.conf**. Universal Broker refreshes the following configuration options: • MESSAGE_LANGUAGE • RUNNING_MAX
Step 2	Read all component definitions found in the component definition directory. Universal Broker replaces all component definitions with the newly read component definitions. New component definitions are added and deleted component definitions are removed.
Step 3	Read the Universal Access Control List configuration file **uacl.conf**. Universal Broker replaces its UACL entries with the newly read entries.

Components

Universal Control

Refreshing a Component from UNIX

- Refreshing a Component via Universal Control from UNIX
  - Command Line Options
  - Components

Refreshing a Component via Universal Control from UNIX

This example refreshes a Universal Event Monitor Server (uems) component on a remote system.

```
uctl -refresh uems -cmdid "ABC-dallas" -host dallas -userid joe -pwd akkSdiq
```

This example refreshes a Universal Automation Center Agent (uag) component on a remote system.

```
uctl -refresh uag -cmdid "ABC-dallas" -host dallas -userid joe -pwd akkSdiq
```

Command Line Options

The command line options used in this example are:

Option	Description
-refresh	Type of component to refresh on the remote system.
-cmdid	Assigns a command identifier of **"ABC-dallas"** to the started component.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the Universal Control Server process.
-pwd	Password for the user ID.

Components

Universal Control

Refreshing Universal Broker from IBM i

- Refreshing Universal Broker via Universal Control from IBM i
  - Command Line Options
  - Universal Broker Actions
  - Components

Refreshing Universal Broker via Universal Control from IBM i

This example refreshes a component on a remote system.

```
STRUCT REFRESH(*yes) HOST(dallas) USERID(joe) PWD(akkSdiq)
```

Command Line Options

The command line options used in this example are:

Option	Description
REFRESH	Instruction to refresh Universal Broker on the remote system.
HOST	Directs the command to a computer with a host name of **dallas**.
USERID	Remote user ID with which to execute the Universal Control Server process.
PWD	Password for the user ID.

Universal Broker Actions

The REFRESH command directs Universal Broker to take the following actions:

Step 1	Read its configuration file **UNVCONF** and member **UBROKER**.
Step 2	Read all component definitions found in the component definition file, **UNVPRD520** / **UNVCOMP**. Universal Broker replaces all component definitions with the newly read component definitions. New component definitions are added and deleted component definitions are removed.
Step 3	Read the Universal Access Control List configuration file **UNVCONF** and member **UACL**. Universal Broker replaces its UACL entries with the newly read entries.

Components

Universal Control

Refreshing a Component from IBM i

- Refreshing a Component via Universal Control from IBM i
  - Command Line Options
  - Components

Refreshing a Component via Universal Control from IBM i

This example refreshes a Universal Event Monitor Server (uems) component on a remote system.

```
STRUCT REFRESH(*yes) RFSHCMPNM(uems) CMDID('ABC-dallas') HOST(dallas) USERID
(joe) PWD(akkSdiq)
```

This example refreshes a Universal Automation Center Agent (uag) component on a remote system.

```
STRUCT REFRESH(*yes) RFSHCMPNM(uag) CMDID('ABC-dallas') HOST(dallas) USERID
(joe) PWD(akkSdiq)
```

Command Line Options

The command line options used in this example are:

Option	Description
REFRESH	Specification for whether or not to refresh.
RFSHCMPNM	Type of component to refresh on the remote system.
CMDID	Assigns a command identifier of **'ABC-dallas'** to the started component.
HOST	Directs the command to a computer with a host name of **dallas**.
USERID	Remote user ID with which to execute the Universal Control Server process.
PWD	Password for the user ID.

Components

Universal Control

Merging Configuration Options

- Overview
- Merging during Upgrades
- Merging at any Time
- Examples

Overview

The Universal Products Install Merge (UPIMERGE) utility merges options and values from one Universal Agent component configuration file or component definition file with another.

Merging during Upgrades

UPIMERGE runs automatically during 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 Universal Agent features.

Merging at any Time

The Universal Agent (UNIX and Windows) and Universal Enterprise Controller (Windows only) distribution packages also install UPIMERGE. This makes UPIMERGE available at any time for recovering archived options and values and merging them with the most recent options and values.

When used to update a Universal Agent configuration or component definition file, UPIMERGE must run with a user account that has write access to the output file. This typically means administrative access (that is, root on UNIX, Administrator on Windows).

Examples

The following pages provide examples of how configuration files can be merged:

- Files Used in UPI Merge Examples
- Merge Configuration Files Using Program Defaults
- Merge Configuration Files Introducing New Options
- Merge Configuration Files Using Installation-Dependent Values

These examples illustrate the merging of Universal Agent (for Windows or UNIX) components' configuration options using the Universal Products Install Merge (UPI) component.



The information on these pages also is located in Universal Agent 7.3.x User Guide.pdf.

Files Used in UPI Merge Examples

- Files Used in Examples
  - Universal Agent Configuration File Sample (infile.txt)
  - Universal Agent Configuration File Sample (outfile.txt)

Files Used in Examples

The examples in this section demonstrate the expected results when Universal Products Install Merge is executed using two configuration files with the contents identified in the following tables.

> **Note**
> Although these examples show Windows path names, the Universal Install Merge behavior demonstrated also applies to UNIX systems.

Universal Agent Configuration File Sample (infile.txt)

The following table identifies the contents of **infile.txt**, a sample file in the Universal Agent standard keyword / value configuration file format.

For the examples in this section, **infile.txt** could represent an existing or archived configuration file, or a work file used to introduce and distribute configuration values across one or more target systems.

Keyword	Value
installation_directory	"C:\Program Files\Universal\UCmdMgr"
message_level	info
#host	some.remote.host
port	7850
license_product	"UNIVERSAL COMMAND MANAGER"
license_customer	"STONEBRANCH, INC."
license_type	DEMO
license_expiration_date	2012.12.21
license_nt_servers	1
license_key	078B-E180-64E6-3016-EA20-0CF4-58F9-B301 *

* This license key is for demonstration purposes only. It is not a valid license key.

Universal Agent Configuration File Sample (outfile.txt)

The following table identifies the contents of **outfile.txt**, another sample file in Universal Agent standard keyword / value configuration file format.

For the examples in this section, **outfile.txt** might represent a default configuration file that is delivered during product installation, or an existing production configuration file that needs to be updated with values from **infile.txt**.

Keyword	Value
port	7887

activity_monitoring	yes
event_generation	*,x100

Merge Configuration Files Using Program Defaults

- Merge Configuration Files Using Program Defaults
  - Command Line Options
  - Merged File Contents
  - Components

Merge Configuration Files Using Program Defaults

The following figure illustrates the command line used to merge configuration options from **`infile.txt`** into **`outfile.txt`**.

In this example, UPIMERGE executes using program defaults.

```
upimerge -dest outfile.txt -source infile.txt
```

Command Line Options

The command line options used in this example are:

Option	Description
-dest	Name of a file used to store the result of the merge.
-source	Name of a file used as input to the merge. If this option is omitted, UPIMERGE assumes input is redirected via stdin.

Merged File Contents

The following table identifies the contents of **`outfile.txt`** after UPIMERGE completes.

To obtain this result, UPIMERGE added options from **`infile.txt`** that did not exist in **`outfile.txt`** (that is, installation_directory, message_level, license_key, and so on). It also preserved the value for the **port** option by replacing the 7887 value with the currently defined 7850.

UPIMERGE also dropped the commented **host** option from **`infile.txt`**. UPIMERGE ignores any comments in the input file, because merging those lines into the output file would have no effect on the application's behavior.

Finally, UPIMERGE commented out the **activity_monitoring** and **event_generation** options introduced by **`outfile.txt`**. UPIMERGE cannot distinguish between options for new features and new values for existing options. To prevent the introduction of a new value into an application currently running with application-defined defaults, UPIMERGE's default response is to comment out any option in the output file with no match in the input file.

Keyword	Value
installation_directory	"C:\Program Files\Universal\UCmdMgr"
message_level	info
port	7850
license_product	"UNIVERSAL COMMAND MANAGER"
license_customer	"STONEBRANCH, INC."
license_type	DEMO
license_expiration_date	2012.12.21
license_nt_servers	1
license_key	078B-E180-64E6-3016-EA20-0CF4-58F9-B301
#activity_monitoring	yes
#event_generation	*,x100

- Merge Configuration Files Introducing New Options
  - Command Line Options
  - Merged File Contents
  - Components

Merge Configuration Files Introducing New Options

The following figure illustrates the command line used to merge configuration options from **`infile.txt`** into **`outfile.txt`**.

In this example, UPIMERGE changes its default behavior, and introduces new values for the **activity_monitoring** and **event_generation** options by not commenting them out in the merged file.

```
upimerge -dest outfile.txt -source infile.txt -keep_nomatch yes
```

Command Line Options

The command line options used in this example are:

Option	Description
-dest	Name of a file used to store the result of the merge.
-source	Name of a file used as input to the merge. If this option is omitted, UPIMERGE assumes input is redirected via stdin.
-keep_nomatch	Controls merge behavior when an option in -dest has no match in -source.

Merged File Contents

The following table identifies the contents of **`outfile.txt`** after UPIMERGE completes.

The result is almost identical to the example shown in Merge Configuration Files Using Program Defaults. Executing UPIMERGE with **-keep_nomatch** set to **yes** enables the **activity_monitoring** and **event_generation** options in the output file.

Keyword	Value
installation_directory	"C:\Program Files\Universal\UCmdMgr"
message_level	info
port	7850
license_product	"UNIVERSAL COMMAND MANAGER"
license_customer	"STONEBRANCH, INC."
license_type	DEMO
license_expiration_date	2012.12.21
license_nt_servers	1
license_key	078B-E180-64E6-3016-EA20-0CF4-58F9-B301
activity_monitoring	yes
event_generation	*,x100

- Merge Configuration Files Using Installation-Dependent Values
  - Command Line Options
  - Merged File Contents
  - Components

Merge Configuration Files Using Installation-Dependent Values

The following figure illustrates the command line used to merge configuration options from **`infile.txt`** into **`outfile.txt`**. In this example, UPIMERGE applies logic specific to a particular configuration file, and updates any references to locations that depend on the installed location of that Universal Agent application.

```
upimerge -dest outfile.txt -source infile.txt -cfgtype ucmd -installdir "D:
\Program Files\Universal\UCmdMgr"
```

Command Line Options

The command line options used in this example are:

Option	Description
-dest	Name of a file used to store the result of the merge.
-source	Name of a file used as input to the merge. If this option is omitted, UPIMERGE assumes input is redirected via stdin.
-cfgtype	Notifies UPIMERGE that -source is a configuration file that contains settings for the specified Universal Agent application.
-installdir	Primary location in which the Universal Agent application identified by -cfgtype resides.

Merged File Contents

The following table identifies the contents of **`outfile.txt`** after UPIMERGE completes. The result is almost identical to the example shown in Merge Configuration Files Using Program Defaults, except for the value of the **-installdir** option.

Even though **`infile.txt`** contained a value for*-installdir*, UPIMERGE interpreted that value as the application's current location. UPIMERGE then updated any values in **`outfile.txt`** (executing logic based on the specified **-cfgtype**) that depend on the installed location.

This example might be useful in a situation where it is necessary to recover configuration settings from an archived file, but the application no longer resides in the directory specified in the archive file. This is the logic that UPIMERGE uses during a Universal Agent installation to ensure that installation-dependent locations are always correct.

Keyword	Value
installation_directory	"D:\Program Files\Universal\UCmdMgr"
message_level	info
Port	7850
license_product	"UNIVERSAL COMMAND MANAGER"
license_customer	"STONEBRANCH, INC."
license_type	DEMO
license_expiration_date	2012.12.21
license_nt_servers	1
license_key	078B-E180-64E6-3016-EA20-0CF4-58F9-B301
#activity_monitoring	yes
#event_generation	*,x100

Components

Universal Products Install Merge

Configuration Options

Configuration Options

The following configuration options are available for Universal Agent components:

OMS Server Configuration Options

Universal Automation Center Agent Configuration Options

Universal Broker Configuration Options

Universal Command Manager Configuration Options

Universal Command Server Configuration Options

Universal Command Agent for SOA Configuration Options

Universal Connector for SAP Configuration Options

Universal Data Mover Manager Configuration Options

Universal Data Mover Server Configuration Options

Universal Enterprise Controller Configuration Options

UECLoad Configuration Options

Universal Event Monitor Manager Configuration Options

Universal Event Monitor Server Configuration Options

UEMLoad Configuration Options

Universal Certificate Configuration Options

Universal Control Manager Configuration Options

Universal Control Server Configuration Options

Universal Copy Configuration Options

Universal Database Dump Configuration Options

Universal Database Load Configuration Options

Universal Display Log File Configuration Options

Universal Encrypt Configuration Options

Universal Event Log Dump Configuration Options

Universal FTP CLient Configuration Options

Universal Message Translator Configuration Options

Universal Products Install Merge Configuration Options

Universal Query Configuration Options

Universal Spool List Configuration Options

Universal Spool Remove Configuration Options

Universal Submit Job Configuration Options

Universal Write-to-Operator Configuration Options

Component Management

Component Management

Component Management information for Universal Agent is comprised of:

- Component Definition
- Component Definition Options
- Starting and Stopping Components
- Starting and Stopping Components Examples
- Maintaining Universal Broker Definitions in UEC Database

Component Definition

Overview

Each Universal Agent server component - Universal Automation Center Agent (UAG), Universal Command Server, Universal Data Mover Server, Universal Event Monitor Server, Universal Control Server, Universal Application Container, and Universal Message Service (OMS) - has a component definition.

The Component Definition is a text file of options that defines component-specific information required by the Universal Broker.

Each Component Definition defines the following type of information:

- Component type (for Universal Event Monitor Servers only).
- Component name.
- Component command name.
- Component configuration file name.
- Component working directory path.
- Number of component instances that can run simultaneously.
- Specification for whether or not the component starts automatically when the Universal Broker starts.

For information on the options that comprise each Component Definition, see:

- Universal Automation Center Agent
- Universal Command
- Universal Data Mover
- Universal Event Monitor
- Universal Control
- Universal Application Container
- Universal Message Service (OMS)

Universal Event Monitor Component Definition

The Component Definition for a Universal Event Monitor Server defines whether it is a demand-driven or an event-driven server. Among other factors, this determines how the server is started (see Starting and Stopping Agent Components).

For a complete explanation of the difference between demand-driven and event-driven Universal Event Monitor Servers, see UEM Servers - Demand-Driven vs. Event-Driven.

Component Definition Options

Component Definition Options

The following component definition options are available for Universal Agent components:

Universal Broker Component Definition Options

Universal Automation Center Agent Component Definition Options

Universal Command Component Definition Options

Universal Data Mover Component Definition Options

UAC Server Component Definition Options

Universal Event Monitor Component Definition Options

Universal Control Component Definition Options

Starting and Stopping Agent Components

- Starting Components
  - Starting Manually
  - Starting via Manager
  - Starting Automatically
  - Starting via Universal Control
- Stopping Components

Starting Components

There are four ways in which Universal Agent components are started.

Starting Manually

The following components are started manually and run in the background until they are stopped manually:

- Universal Broker
- Universal Enterprise Controller

(See Starting and Stopping Agent Components - Examples.)

Starting via Manager

The following components are started on demand (that is, via their Managers) and run until the specified task has completed, then stop automatically.

- Universal Command Server
- Universal Control Server
- Universal Event Monitor Server (demand-driven)

Starting Automatically

The following components are auto-start components; that is, they start automatically when the Universal Broker starts and run until they are stopped manually:

- Universal Application Container Server
- Universal Event Monitor Server (event-driven)
- Universal Automation Center Agent (UAG)
- Universal Message Service (OMS)

> **Note**
> The AUTOMATICALLY_START component definition option for Universal Event Monitor Server also can specify that an event-driven server is not started automatically (see #Starting via Universal Control, below).
>
> The AUTOMATICALLY_START component definition option for OMS also can specify that it is not started automatically.

Starting via Universal Control

Universal Control can start Server components, via the Universal Control START_CMD option, that do not require interaction with a Manager. Currently, only three Universal Agent components can be started via Universal Control:

- Universal Event Monitor Server (event-driven)
- Universal Automation Center Agent (UAG)
- Universal Message Service (OMS)

(See Starting and Stopping Agent Components - Examples.)

Stopping Components

Any Universal Agent Server component can stopped via the Universal Control STOP_CMD option.

Authorized users also are able to use the I-Activity Monitor, a Universal Enterprise Controller (UEC) client application, to stop running any Universal Agent Server component (if it is a component of an Agent being polled by UEC).

Starting and Stopping Agent Components - Examples

Starting and Stopping Universal Broker Examples

- Starting and Stopping Universal Broker - z/OS
- Starting Universal Broker - Windows
- Starting Universal Broker - UNIX
- Starting, Ending, Working with Universal Broker - IBM i

Starting and Stopping Universal Enterprise Controller Examples

- Starting and Stopping Universal Enterprise Controller - z/OS
- Starting and Stopping Universal Enterprise Controller - Windows

Starting and Stopping Components via Universal Control Examples

> **Note**
> Currently, only Universal Event Monitor Servers and Universal Automation Center Agent can be started by Universal Control.

The examples assume that Universal Control Server is installed on a remote system named **dallas**. The user ID and password used in the examples must be changed to a valid user ID and password for the remote system.

Links to detailed technical information on appropriate Universal Agent components are provided for each example.

- Starting a z/OS Component via Universal Control
- Stopping a z/OS Component via Universal Control
- Starting a Windows Component via Universal Control
- Stopping a Windows Component via Universal Control
- Starting a UNIX Component via Universal Control
- Stopping a UNIX Component via Universal Control
- Starting an IBM i Component via Universal Control
- Stopping an IBM i Component via Universal Control

> **Note**
> The IBM i examples reference the IBM i commands by their untagged names. If you are using commands with tagged names to run Universal Broker and Universal Control, substitute the tagged names for these untagged names. (For information on tagged names, see UCHGRLS (Change Release Tag) Program.)

Starting and Stopping Universal Broker - z/OS

- Overview
- Start Universal Broker
- Stop Universal Broker
  - Stop Universal Broker (Failover)

Overview

Universal Broker for z/OS executes as a started task.

The UBROKER program utilizes the z/OS UNIX System Services environment.

Start Universal Broker

To start Universal Broker, execute the **START** console command:

```
START UBROKER[,UPARM='options']
```

Stop Universal Broker

To stop Universal Broker, execute the **STOP** console command:

```
STOP UBROKER
```

Stop Universal Broker (Failover)

```
F <ubroker>,APPL=SHUTDOWN, [ FAILOVER [ ,<sysname> ] | NOFAILOVER ]
```

When issued against a Secondary agent	This command behaves like the STOP console command.
When issued against a Primary agent	This command shuts down the Broker (and agent) while controlling the Sysplex failover behaviour:
When issued without the FAILOVER or NOFAILOVER parameter	Failover will behave as configured by the automatic_failover parameter in UAGCFG00.
When FAILOVER Is specified	An available Secondary agent will take over as Primary, regardless of how failover is configured.

	When the optional <sysname> is specified, the agent running on the designated z/OS system will take over as Primary agent regardless of how fail over is configured.
When NOFAILOVER Is specified	No Secondary agent will take over as Primary, regardless of how failover is configured.

Note
Behaviour of the STOP console command with failover is identical to this F <ubroker>,APPL=SHUTDOWN command with no other parameters.

Starting Universal Broker - Windows

- Overview
- Console Application
  - Console Security
- Windows Service
  - Service Security
  - Required File System Permissions
  - Executing the Broker Service With a Domain Account
  - Options

Overview

Universal Broker can be executed in two different environments:

- Console application
- Windows service

Console Application

The **ubroker** command starts Universal Broker as a console application.

Enter **ubroker** either from the:

- Command Prompt window
- **Run** dialog (Select **Run...** from the Windows **Start** menu.)

Console Security

Universal Broker inherits its user account from the user that starts it. The Broker itself does not require any additional permissions or rights other than the default ones granted to the Windows group user.

However, components started by the Broker also run with the same user account as the Broker. Some components may require permissions or rights other than those granted to the user account that started the Broker.

For additional information regarding the security requirements of Universal Broker and all Universal Agent components, see Universal Agent Security.

Windows Service

Universal Broker is installed as a Windows service that starts automatically when the system is started. Windows provides a utility called **Services** that is used to interact with and manage all installed services. **Services** is an item in the Administrative Tools program group, which is accessible from the Control Panel.

Service Security

The Universal Broker service can be configured to execute with the Local System account or with a specially configured Administrative account. The Local System account automatically provides the permissions necessary to execute the Broker.

An administrative account must have the following privileges to execute the Broker:

- Act as part of the operating system

- Adjust memory quotas for a process
- Bypass traverse checking
- Debug programs
- Log on as a service
- Impersonate a client after authentication
- Increase scheduling priority
- Replace a process level token
- Take ownership of files and other objects

To restrict interactive access by the account to the system, we also recommend adding the following policies:

- Deny log on as batch job
- Deny log on locally
- Deny log on through Terminal Services

Any existing Administrative account may be configured as described above to execute the Broker. The Universal Agent install also provides the ability to create and configure an Administrative account with the privileges above.

Configuring the Broker to run with an Administrative account not only allows the service to execute with just the privileges it needs, it also enables the Broker service to access network resources it would not have visibility to while executing as Local System.

Required File System Permissions

It may be necessary to update the Broker account's access to the Universal Agent installed directories and files. If the product is installed to its default location under the Program Files directory, the local Administrative account used to execute the Broker (such as the default **UBrokerService** account) will likely get the file system access it needs via permissions inherited from parent directories.

However, if the application is installed to a location outside of the Program Files path - or a domain account is used to execute the Broker Service - the required file system permissions may need to be added after the install.

The recommended approach is to grant the Broker service account Full Control of the following directories, making sure that the permissions are propagated to all sub-directories and files:

- **.\Universal** install directory.
- **%ALLUSERSPROFILE%\Application Data\Universal** directory, which is the parent directory of the **.\conf** and **.\comp** directories in which the configuration files and component definition files reside, respectively.

Full control is recommended because of the varied requirements and configurations possible with the Universal Agent components. However, should you desire a more precise configuration, the Broker user only requires Read/Execute permissions for the following directories, along with their sub-directories and files:

- **.\Universal\nls**
- **.\Universal\UCmdMgr**
- **.\Universal\UCtlMgr**
- **.\Universal\UDMMgr**
- **.\Universal\UEId**
- **.\Universal\UEMMgr**
- **.\Universal\UPIMerge**
- **.\Universal\UQuery**
- **.\Universal\USpool**

> **Note**
> The Universal Agent installation itself does not set the required file permissions for the Broker user. It only relies on permissions inherited from parent directories.

Executing the Broker Service With a Domain Account

The Universal Broker service may be configured to run with a Windows domain account.

To do so, verify the following before starting the installation (the Universal Agent install will not configure a domain account):

- Account already exists.
- Account belongs to the Administrators group.

  > **Note**
  > Depending on your environment, it may be necessary to add this account to the Domain Admins group. This will ensure the account has sufficient access to domain resources and is recognized as a true administrative account on all domain member systems that run the Universal Broker service as that account.

- Account has the privileges and file system permissions listed above.

Options

Option Name	Description
COMMAND	Command to execute for the ubrokerd daemon: • **start** Start the ubrokerd daemon. • **stop** Stop the ubrokerd daemon. • **restart** Stop/start the ubrokerd daemon. • **status** Query the ubrokerd daemon status.
-uag_agent_clusters	Overrides the AGENT_CLUSTERS option in the UAG configuration file. A list of one or more comma-separated agent clusters defined in the Universal Controller that the Agent should join.
-uag_transient	Overrides the TRANSIENT option in the UAG Server configuration file. • If **yes**, UAG Server will register with the Controller as a transient agent, and un-register from the Controller automatically when going offline. • If **no**, UAG Server will NOT enable the transient option. If this option is omitted, the value specified for the TRANSIENT option in the UAG Server config file controls start-up behavior.
-uag_netname	Overrides the network ID of Universal Automation Center Agent (UAG) in the NETNAME UAG configuration option. UAG Server will use this ID when connecting to a Universal Controller. If the NETNAME value is OPSAUTOCONF (the default), and the UAG Server already has connected to a Universal Controller, the qname value that holds the UAG Server's assigned netname must be deleted before it can be overridden by this option.
-uag_oms_servers	Overrides the values in the OMS_SERVERS UAG configuration option. A list of one or more OMS Servers to which this Agent will connect.
-uag_autostart	Overrides the AUTOMATICALLY_START option in the UAG Server component definition. • If **yes**, OMS Server will start automatically at Universal Broker start-up, regardless of the value specified in the component definition file. • If **no**, OMS Server will not start when the Universal Broker starts, regardless of the value specified in the component definition file. If this option is omitted, the value specified in AUTOMATICALLY_START controls start-up behavior.
-uag_extension_accept_list	Overrides the EXTENSION_ACCEPT_LIST option in the UAG configuration file. Specifies a list of Universal Extensions that the Agent will accept via auto-deployment from Universal Controller. The list consists of one or more comma-separated Extension names. • A single value of ***** indicates that all extensions are accepted. • A single value of **none** indicates that no extensions are accepted.
-uag_extension_cancel_timeout	Overrides the EXTENSION_CANCEL_TIMEOUT option in the UAG configuration file. Specifies the amount of time that an Extension process will be allowed to run following a Cancel message being received from the Controller. If the Extension process is still running after the extension_cancel_timeout expires, UAG will forcibly terminate the process. The format of the value is nnnn[s|m|h|d], where nnnn is a numeric value and [s|m|h|d] is one of the following optional unit specifiers: • s - seconds (default) • m - minutes

	• h - hours • d - days
-uag_extension_deploy_on_registration	Overrides the EXTENSION_DEPLOY_ON_REGISTRATION option in the UAG configuration file. Controls Extension deployment behavior from Universal Controller. • If **yes**, the Controller will preemptively deploy all extensions acceptable by UAG. • If **no**, the Controller will only send Extension modules as needed (on demand).
-uag_extension_python_list	Overrides the EXTENSION_PYTHON_LIST option in the UAG configuration file. Specifies a comma-separated list of zero or more Python locations. Each item in the list is expected to contain a complete path to a Python executable.
-oms_autostart	Overrides the AUTOMATICALLY_START option in the OMS Server component definition file. • If **yes**, OMS Server will start automatically at Universal Broker start-up, regardless of the value specified in the component definition file. • If **no**, OMS Server will not start when the Universal Broker starts, regardless of the value specified in the component definition file. If this option is omitted, the value specified in AUTOMATICALLY_START controls start-up behavior.
-uem_autostart	Overrides the AUTOMATICALLY_START option in the UEM definition file. • If **yes**, UEM Server will start automatically at Universal Broker startup, regardless of the value specified in the component definition file. • If **no**, UEM Server will not start when the Universal Broker starts, regardless of the value specified in the component definition file. If this option is omitted, the value specified in AUTOMATICALLY_START controls start-up behavior.
h	Display program usage.

Starting Universal Broker - UNIX

- Starting Universal Broker for UNIX
- Console Application
  - Console Security
- Daemon
  - Usage
  - Options
  - Daemon Security

Starting Universal Broker for UNIX

Universal Broker can be executed in two different environments:

- Console Application
- Daemon

Differences between the environments are described in the following sections.

Only one instance of the Universal Broker can execute at any one time. A PID file is used to help ensure that there is only one active instance; it is a locking mechanism that prevents the execution of a second Broker. The PID file, **`ubroker.pid`**, is created in directory **`/var/opt/universal`** by default. If the PID file is in the PID directory, it is assumed that a Broker instance is executing.

Console Application

The **ubroker** command starts Universal Broker as a console application.

Console Security

Universal Broker runs with the same user ID as the user who starts it; it does not require superuser rights. Universal Broker only requires access to its installation directory and files, which often are created by the superuser account when the product is installed.

However, components started by Universal Broker also run with the same user ID as Universal Broker. Some of these components *may* require superuser rights.

See Universal Agent Security for details on their security requirements for specific Universal Agent components.

Daemon

Universal Broker can run as a UNIX daemon process. This is the preferred method of running the Broker.

A daemon start-up script, **ubrokerd**, is provided to manage the starting and stopping of the Broker daemon. The start-up script utilizes the PID file to ensure that only one instance of the Broker is executing at any one time. For this reason, the start-up script should be used to start and stop the Broker.

Note
Although they have the same name, the Broker daemon start-up script (**ubrokerd**) should not be confused with the actual Broker daemon program file.

- Daemon *start-up script* is installed in the primary Broker directory (that is, **`./universal/ubroker`**).
- Daemon *program file* is installed in the Broker's **`bin`** directory (that is, **`./universal/ubroker/bin`**).

```
ubrokerd { start | stop | status | restart }
```

The following table describes the command line arguments to the Universal Broker daemon start-up script.

Command	Description
start	Starts the Universal Broker daemon. Only one instance of Universal Broker can run at any given time, so if the Broker already is running, the command fails and the script returns.
stop	Stops the Universal Broker daemon. If the Broker daemon is not running, the script simply returns.
status	Returns the status of the Universal Broker daemon, either *running* or *stopped*. If the daemon is running, the script displays its process ID.
restart	Performs a **stop** request followed by a **start** request.

Usage

ubrokerd command [-uag_agent_clusters list] [-uag_netname name] [-uag_oms_servers port1@ip1[;...portn@ipn] [-uag_autostart opt] [-oms_autostart opt] [-uag_extension_accept_list list] [-uag_extension_cancel_timeout time] [-uag_extension_python_list path1[,pathn]] [-uag_extension_deploy_on_registration opt] [-uag_process_cancel_timeout time] [-uem_autostart opt] [-uag_transient opt] [-h]

Options

Option Name	Description
COMMAND	Command to execute for the ubrokerd daemon: • **start** Start the ubrokerd daemon. • **stop** Stop the ubrokerd daemon. • **restart** Stop/start the ubrokerd daemon. • **status** Query the ubrokerd daemon status.
-uag_agent_clusters	Overrides the AGENT_CLUSTERS option in the UAG configuration file. A list of one or more comma-separated agent clusters defined in the Universal Controller that the Agent should join.
-uag_transient	Overrides the TRANSIENT option in the UAG Server configuration file. • If **yes**, UAG Server will register with the Controller as a transient agent, and un-register from the Controller automatically when going offline. • If **no**, UAG Server will NOT enable the transient option.

	If this option is omitted, the value specified for the TRANSIENT option in the UAG Server config file controls start-up behavior.
-uag_netname	Overrides the network ID of Universal Automation Center Agent (UAG) in the NETNAME UAG configuration option. UAG Server will use this ID when connecting to a Universal Controller. If the NETNAME value is OPSAUTOCONF (the default), and the UAG Server already has connected to a Universal Controller, the qname value that holds the UAG Server's assigned netname must be deleted before it can be overridden by this option.
-uag_oms_servers	Overrides the values in the OMS_SERVERS UAG configuration option. A list of one or more OMS Servers to which this Agent will connect.
-uag_autostart	Overrides the AUTOMATICALLY_START option in the UAG Server component definition. • If **yes**, OMS Server will start automatically at Universal Broker start-up, regardless of the value specified in the component definition file. • If **no**, OMS Server will not start when the Universal Broker starts, regardless of the value specified in the component definition file. If this option is omitted, the value specified in AUTOMATICALLY_START controls start-up behavior.
-oms_autostart	Overrides the AUTOMATICALLY_START option in the OMS Server component definition file. • If **yes**, OMS Server will start automatically at Universal Broker start-up, regardless of the value specified in the component definition file. • If **no**, OMS Server will not start when the Universal Broker starts, regardless of the value specified in the component definition file. If this option is omitted, the value specified in AUTOMATICALLY_START controls start-up behavior.
-uag_extension_accept_list	Overrides the EXTENSION_ACCEPT_LIST option in the UAG configuration file. Specifies a list of Universal Extensions that the Agent will accept via auto-deployment from Universal Controller. The list consists of one or more comma-separated Extension names. • A single value of ***** indicates that all extensions are accepted. • A single value of **none** indicates that no extensions are accepted.
-uag_extension_cancel_timeout	Overrides the EXTENSION_CANCEL_TIMEOUT option in the UAG configuration file. Specifies the amount of time that an Extension process will be allowed to run following a Cancel message being received from the Controller. If the Extension process is still running after the extension_cancel_timeout expires, UAG will forcibly terminate the process. The format of the value is nnnn[s|m|h|d], where nnnn is a numeric value and [s|m|h|d] is one of the following optional unit specifiers: • s - seconds (default) • m - minutes • h - hours • d - days
-uag_extension_deploy_on_registration	Overrides the EXTENSION_DEPLOY_ON_REGISTRATION option in the UAG configuration file. Controls Extension deployment behavior from Universal Controller. • If **yes**, the Controller will preemptively deploy all extensions acceptable by UAG. • If **no**, the Controller will only send Extension modules as needed (on demand).
-uag_extension_python_list	Overrides the EXTENSION_PYTHON_LIST option in the UAG configuration file. Specifies a comma-separated list of zero or more Python locations. Each item in the list is expected to contain a complete path to a Python executable.
-uag_process_cancel_timeout	Overrides the PROCESS_CANCEL_TIMEOUT option in the UAG configuration file.

	Specifies the amount of time that an OS task process will be allowed to run following a Cancel message being received from the Controller. If the process is still running after the process_cancel_timeout expires, UAG will forcibly terminate the process. The format of the value is nnnn[s|m|h|d], where nnnn is a numeric value and [s|m|h|d] is one of the following optional unit specifiers: • s - seconds (default) • m - minutes • h - hours • d - days
-uem_autostart	Overrides the AUTOMATICALLY_START option in the UEM definition file. • If **yes**, UEM Server will start automatically at Universal Broker startup, regardless of the value specified in the component definition file. • If **no**, UEM Server will not start when the Universal Broker starts, regardless of the value specified in the component definition file. If this option is omitted, the value specified in AUTOMATICALLY_START controls start-up behavior.
h	Display program usage.

Daemon Security

When a daemon is started at system initialization, it is started as user **root**. The root user ID provides sufficient authority for the Broker and any component it may start.

If the daemon is started with a non-root user ID, the environment is the same as if it was started as a console application. (See Console Security, above, for more details.)

Starting, Ending, Working with Universal Broker - IBM i

- Starting, Ending, and Working With Universal Broker for IBM i
- Commands
  - Start Subsystem Command (STRSBS)
  - End Subsystem Command (ENDSBS)
  - Work With Subsystem Command (WRKSBS)

Starting, Ending, and Working With Universal Broker for IBM i

Universal Broker executes within its own IBM i subsystem, named **UNVUBR520**. The **UNVUBR520** subsystem provides a self-contained environment in which Universal Broker can be managed. The **UNVUBR520** subsystem description (object type ***SBSD**) is named **UNVUBR520**.

The **UNVUBR520** subsystem contains several entries that define the subsystem environment. The two most visible are:

- Autostart entry
- Pre-start job entries

The subsystem autostart entry defines what jobs are started automatically when the subsystem is started. The **UNVUBR520** subsystem defines one autostart entry, **UNVUBR520**. The **UBROKER** job executes with the job description **UBROKER** (object type ***JOBD**) and user profile **UNVUBR520** (object type ***USRPRF**). Only one instance of the **UBROKER** job, which runs continuously, can be active at any one time within the context of any one Stonebranch-defined subsystem.

The subsystem pre-start job entries define jobs that are in an initialized state. They are not executing but are ready to accept a request and execute at any time. Pre-starting jobs before they are required improves the overall throughput of the subsystem jobs.

Universal Broker jobs running under **UNVUBR520** use the **UBROKER** job queue and class located in the product installation library. See IBM i Installation - Customization for additional information.

The Universal Command (UCMD) Server jobs log all significant events to the **UBROKER** job log. However, by default, IBM i does not keep job logs unless the job terminates due to an error. As a result, important information relevant to server errors may be discarded when the **UBROKER** job is shut down normally.

To preserve the server-related information, the UBROKER job description specifies Message Logging as *4 0 *MSG*. The **UBROKER** job's job log will be sent automatically to the output queue and printer device designated in the **UBROKER** job description, which is located in the Universal Agent installation library, **UNVPRD520** (by default).

In some very large organizations with heavy **UBROKER** usage, the job log may fill. By default, IBM i jobs are stopped when the job log fills. To ensure continuous **UBROKER** operation, Universal Agent sets the job log to wrap. (See IBM i Installation for additional information.)

Commands

The following O/S commands help manage the **UNVUBR520** subsystem.

Start Subsystem Command (STRSBS)

Starts the Universal Broker subsystem, **UNVUBR520**.

```
STRSBS UNVPRD520/UNVUBR520
```

End Subsystem Command (ENDSBS)

Ends the Universal Broker subsystem, **UNVUBR520**.

```
ENDSBS UNVUBR520
```

Work With Subsystem Command (WRKSBS)

Allows users to work with all active subsystems. Choose the **UNVUBR520** subsystem from the list of subsystems displayed.

```
WRKSBS
```

Starting and Stopping Universal Enterprise Controller - z/OS

- Overview
- Starting UEC
- Stopping UEC
- System MODIFY Command
  - DUMP Command
  - BROKERSTAT Command

Overview

Universal Enterprise Controller (UEC) for z/OS executes as a started task.

Starting UEC

The UEC started task, **UECTLR**, is started with the z/OS START command:

```
*S UECTLR*
```

Stopping UEC

The UEC started task, **UECTLR**, is stopped with the z/OS MODIFY STOP command:

```
*P UECTLR*
```

After the STOP command is issued, UEC may take several seconds to shut down.

> **Note**
> The **UECTLR** started task should run at a high dispatch priority in order to avoid not being dispatched in a timely enough manner to process the agent polling protocol. If **UECTLR** is not dispatched appropriately, the Broker may be reported as timed out when the Broker itself still is operational.

System MODIFY Command

The UEC started task accepts commands via the system MODIFY command. The MODIFY command's **APPL=** parameter is required, since UEC runs as a USS address space.

DUMP Command

The DUMP command directs UEC to produce a Language Environment dump. The dump is written to the **CEEDUMP** ddname. While the dump is being produced, UEC is paused by LE until the dump completes, after which UEC continues processing.

In the following example, the procedure name **UECTLR** is assumed:

```
*F UECTLR,APPL=DUMP*
```

The DUMP command is used for diagnostic purposes. It should be executed only at the request of Stonebranch, Inc.

BROKERSTAT Command

The BROKERSTAT command provides on-demand Broker status alerting. It causes UEC to issue an alert message for all defined Brokers indicating their current internal state.

- Alert UNV1056T (Unable to connect) is issued for Brokers that are down.
- Alert UNV1059T (Broker responding) is issued for Brokers that are up.

The alert message is equivalent to what UEC issued at the time the alert was originally generated.

In the example below, the procedure name **UECTLR** is assumed:

```
*F UECTLR,APPL=BROKERSTAT*
```

Alerts issued on-demand (by BROKERSTAT) are not sent to the I-Activity Monitor client. (When issued under normal processing by UEC, the alerts are sent to I-Activity Monitor.)

Starting and Stopping Universal Enterprise Controller - Windows

Starting / Stopping Universal Enterprise Controller for Windows

Universal Enterprise Controller (UEC) for Windows executes as a service.

By default, UEC for Windows is set to start automatically whenever Windows is booted.

Changes to UEC configuration require it to be stopped and restarted by the Windows Service Control Manager.

To access the Service Control Manager:

Step 1	Click the **Control Panel** on the Windows Start menu.
Step 2	Double-click the **Administrative Tools** icon on the Control Panel window.
Step 3	Double-click the **Services** icon on the Administrative Tools window.
Step 4	On the Services window, select Universal Enterprise Controller in the list of services.
Step 5	In the Action menu, click: 1. Stop, to stop UEC for Windows. 2. Start, to start UEC for Windows.

Starting a z/OS Component via Universal Control

- Starting a z/OS Component via Universal Control
  - SYSIN Options
  - Components

Starting a z/OS Component via Universal Control

This example - located in the Universal Control **SUNVSAMP** library - starts a component on a remote system.

It assumes that Universal Control Server is installed on a remote system named **dallas**. The user ID and password used in the example must be changed to a valid user ID and password for the remote system.

```
//jobname  JOB CLASS=A,MSGCLASS=X
//STEP1    EXEC UCTLPRC
```

```
//SYSIN    DD  *
-start uems -cmdid "UEM-dallas" -host dallas -userid joe -pwd akkSdiq
/*
```

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-start	Name of the component to start on the remote system.
-cmdid	Assigns a command identifier of **"UEM-dallas"** to the started component.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the Universal Control Server process. The started component, in fact, will execute with the Universal Broker's security context.
-pwd	Password for the user ID.

Components

Universal Control

Stopping a z/OS Component via Universal Control

- Stopping a zOS Component via Universal Control
- SYSIN Options
- Components

Stopping a zOS Component via Universal Control

This example - located in the Universal Control **SUNVSAMP** library - stops a component on a remote system.

It assumes that Universal Control Server is installed on a remote system named **dallas**. The user ID and password used in the example must be changed to a valid user ID and password for the remote system.

```
//jobname  JOB CLASS=A,MSGCLASS=X
//STEP1    EXEC UCTLPRC
//SYSIN    DD  *
-stop 999234133 -host dallas -userid joe -pwd akkSdiq
/*
```

The sample JCL is located in member **UCTSAM1**.

The JCL procedure **UCTLPRC** is used to execute the stop request.

The stop request is sent to a remote system named **dallas** for execution.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-stop	Component to stop.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the stop request.
-pwd	Password for the user ID.

Components

Starting a Windows Component via Universal Control

- Starting a Windows Component via Universal Control
  - Command Line Options
  - Components

Starting a Windows Component via Universal Control

This example starts a component on a remote system.

```
uctl -start uems -cmdid "UEM-dallas" -host dallas -userid joe -pwd akkSdiq
```

Command Line Options

The command line options used in this example are:

Option	Description
-start	Name of the component to start on the remote system.
-cmdid	Assigns a command identifier of **"UEM-dallas"** to the started component.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the Universal Control Server process. The started component, in fact, will execute with the Universal Broker's security context.
-pwd	Password for the user ID.

Components

Stopping a Windows Component via Universal Control

- Stopping a Windows Component via Universal Control
  - Command Line Options
  - Components

Stopping a Windows Component via Universal Control

This example stops a component on a remote system.

```
uctl -stop 10739132 -host dallas -userid joe -pwd akkSdiq
```

Command Line Options

The command line options used in this example are:

Option	Description
-stop	Component to stop.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the stop request.
-pwd	Password for the user ID.

Components

Starting a UNIX Component via Universal Control

- Starting a UNIX Component via Universal Control
  - Command Line Options
  - Components

Starting a UNIX Component via Universal Control

This example starts a component on a remote system.

```
uctl -start uems -cmdid "UEM-dallas" -host dallas -userid joe -pwd akkSdiq
```

Command Line Options

The command line options used in this example are:

Option	Description
-start	Name of the component to start on the remote system.
-cmdid	Assigns a command identifier of **"UEM-dallas"** to the started component.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the Universal Control Server process. The started component, in fact, will execute with the Universal Broker's security context.
-pwd	Password for the user ID.

Components

Universal Control

Stopping a UNIX Component via Universal Control

- Stopping a UNIX Component via Universal Control
  - Command Line Options
  - Components

Stopping a UNIX Component via Universal Control

This example stops a component on a remote system.

```
uctl -stop 10739132 -host dallas -userid joe -pwd akkSdiq
```

Command Line Options

The command line options used in this example are:

Option	Description
-stop	Component to stop.
-host	Directs the command to a computer with a host name of **dallas**.
-userid	Remote user ID with which to execute the stop request.
-pwd	Password for the user ID.

Components

Universal Control

Starting an IBM i Component via Universal Control

- Starting an IBM i Component via Universal Control
  - Command Line Options
  - Components

Starting an IBM i Component via Universal Control

This example starts a component on a remote system.

```
STRUCT START(uems) CMDID('UEM-dallas') HOST(dallas) USERID(joe) PWD(akkSdiq)
```

Note
This example references the IBM i command by its untagged name. If you are using commands with tagged names to run Universal Control, substitute the tagged names for the untagged names.

Command Line Options

The command line options used in this example are:

Option	Description
START	Component to start on the remote system.
CMDID	Assigns a command identifier of 'UEM-dallas' to the started component.
HOST	Directs the command to a computer with a host name of **dallas**.
USERID	Remote user ID with which to execute the Universal Control Server process. The started component, in fact, will execute with the Universal Broker's security context.
PWD	Password for the user ID.

Components

Universal Control

Stopping an IBM i Component via Universal Control

- Stopping an IBM i Component via Universal Control
  - Command Line Options
  - Components

Stopping an IBM i Component via Universal Control

This example stops a component on a remote system.

```
STRUCT STOP(10739132) HOST(dallas) USERID(joe) PWD(akkSdiq)
```

Note
This example references the IBM i command by its untagged name. If you are using commands with tagged names to run Universal Control, substitute the tagged names for the untagged names.

Command Line Options

The command line options used in this example are:

Option	Description
STOP	Component on the remote system to stop.

HOST	Directs the command to a computer with a host name of **dallas**.
USERID	Remote user ID with which to execute the stop request. This must match the user ID originally used to start the component.
PWD	Password for the user ID.

Components

Universal Control

Maintaining Universal Broker Definitions in UEC Database

Maintaining Universal Broker Definitions in UEC Database

- Maintaining Broker Definitions in UEC Database - z/OS and Windows
- Maintaining Broker Definitions in UEC Database - z/OS
- Maintaining Broker Definitions in UEC Database - Windows

Note
All of the tasks illustrated on these pages are implemented with use of the UECLoad Utility component.

Maintaining Broker Definitions in UEC Database - zOS and Windows

- List All Defined Universal Brokers
- Export a Specific, Defined Universal Broker
- Export Events
- Delete a Specific, Defined Universal Broker
- Add Specific, Defined Universal Broker via deffile
- Add Existing Universal Brokers to a Broker Group
- Delete Existing Universal Brokers from a Broker Group

List All Defined Universal Brokers

The following illustrates the output of a user-friendly format of the Universal Brokers defined in the UEC database.

```
uecload -port 8778 -userid joe -pwd akkSdiq -list -broker_name "*"
```

Command Line Options

The command line options used in this example are:

Option	Description
-port	TCP/IP port number of the UEC.
-userid	UEC user ID/account with which Brokers will be modified.
-pwd	Password associated with -userid.
-list	Output the described broker definition in a user-friendly format.
-broker_name	"*" specifies all Universal Brokers.

Export a Specific, Defined Universal Broker

The following illustrates the output of a Universal Broker defined in the UEC database in a format suitable for use within a broker definition file.

```
uecload -port 8778 -userid joe -pwd akkSdiq -level audit -export -
broker_name mybroker1
```

Command Line Options

The command line options used in this example are:

Option	Description
-port	TCP/IP port number of the UEC.
-userid	UEC user ID/account with which Brokers will be modified.
-pwd	Password associated with -userid.
-level	Level of messages written.
-export	Output the described broker definition in a format to be used by a broker definition file.
-broker_name	Unique name of the defined Universal Broker.

Export Events

The following illustrates the export of an events file into CSV format.

```
uecload -port 8778 -userid joe -pwd akkSdiq -level audit -export EVENTS -
stime "*-5" -etime "*" -format CSV -deffile events.csv
```

Note
The double quotation marks (") are required only with UNIX.

Command Line Options

The command line options used in this example are:

Option	Description
-port	TCP/IP port number of the UEC.
-userid	UEC user ID/account with which Brokers will be modified.
-pwd	Password associated with -userid.
-level	Level of messages written.
-export	Output the described broker definition in a format to be used by a broker definition file.
-stime	Start time of exported data.
-etime	End time of exported data.
-format	Format of the output from the -export EVENTS action.
-deffile	File containing multiple broker definitions to be added or deleted in the UEC database.

Delete a Specific, Defined Universal Broker

The following figure illustrates the deletion of a Universal Broker defined in the UEC database. Specifically, Universal Broker **mybroker1** is deleted from use of UEC.

```
uecload -port 8778 -userid joe -pwd akkSdiq -level audit -delete -
broker_name mybroker1
```

Command Line Options

The command line options used in this example are:

Option	Description
-port	TCP/IP port number of the UEC.
-userid	UEC user ID/account with which Brokers will be modified.

-pwd	Password associated with -userid.
-level	Level of messages written.
-delete	Delete Agent definitions from UEC.
-broker_name	Unique name of the defined Universal Broker.

Add Specific, Defined Universal Broker via deffile

The following figure illustrates the addition of a group of Universal Broker definitions specified within a definition file in the UEC database. The name **sample_deffile** represents the name of the created file.

```
uecload -port 8778 -userid joe -pwd akkSdiq -level audit -add -deffile
sample_deffile
```

Command Line Options

The command line options used in this example are:

Option	Description
-port	TCP/IP port number of the UEC.
-userid	UEC user ID/account with which Brokers will be modified.
-pwd	Password associated with -userid.
-level	Level of messages written.
-add	Add Agent definitions to UEC.
-deffile	File containing multiple broker definitions to be added or deleted in the UEC database.

Definition File

The following figure is the definition file to be used for this example.

```
<BROKERDEF>
broker_name mybroker1
broker_host localhost
broker_port 7887
broker_desc "This is a description of broker1."
groups "Group 1, Group 2,Group  3"
<BROKERDEF>
<BROKERDEF>
broker_name mybroker2
broker_host 127.0.0.1
broker_port 7887
broker_desc "This is a description of broker2."
groups "Group 1, Group 2, Group 3"
<BROKERDEF>
<BROKERDEF>
broker_name mybroker3
broker_host 10.20.30.40
broker_port 7887
broker_desc "This is a description of broker3."
groups "Group 1, Group 2, Group 3"
<BROKERDEF>
```

Add Existing Universal Brokers to a Broker Group

The following illustrates the addition of existing Universal Brokers to a Broker group.

```
uecload -port 8778 -userid joe -pwd akkSdiq -add -deffile brokers -groups
"Test 1, Test 2, Test 3"
```

Command Line Options

The command line options used in this example are:

Option	Description
-port	TCP/IP port number of the UEC.
-userid	UEC user ID/account with which Brokers will be modified.
-pwd	Password associated with -userid.
-add	Add Agent definitions to specified group(s).
-deffile	File containing multiple broker definitions to be added or deleted in the UEC database.
-groups	Group(s) in which the defined Universal Broker is a member. The Universal Broker will be added to the Group(s).

Delete Existing Universal Brokers from a Broker Group

The following illustrates the deletion of existing Universal Brokers from a Broker group.

```
uecload -port 8778 -userid joe -pwd akkSdiq -delete -deffile brokers -groups
"Test 2, Test 3"
```

Command Line Options

The command line options used in this example are:

Option	Description
-port	TCP/IP port number of the UEC.
-userid	UEC user ID/account with which Brokers will be modified.
-pwd	Password associated with -userid.
-delete	Delete Agent definitions from specified group(s).
-deffile	File containing multiple broker definitions to be added or deleted in the UEC database.
-groups	Group(s) in which the defined Universal Broker is a member. The Universal Broker will be added to the Group(s).

Maintaining Broker Definitions in UEC Database - zOS

- Export Events into ARC Format for z/OS
  - SYSIN Options
- Retrieve Archived File and Export into XML for z/OS
  - SYSIN Options

Export Events into ARC Format for z/OS

The following figure illustrates the export of events into an ARC format file on z/OS.

```
//STEP1           EXEC PGM=UECLOAD,PARM='ENVAR(TZ=EST5EDT)/'
//STEPLIB  DD  DISP=SHR,DSN=#HLQ.UNV.SUNVLOAD
//*
//UNVCONF  DD  DISP=SHR,DSN=#HLQ.UNV.UNVCONF(UECCFG00)
//*
//UNVTRACE DD  SYSOUT=*
```

```
//ARCFILE  DD  DSN=APP.UEC.ARCH,
//             DISP=(,CATLG),UNIT=3390,VOL=SER=STG001,
//             SPACE=(CYL,(5,5)),
//             DCB=(RECFM=FB,LRECL=200,BLKSIZE=8000)
//SYSPRINT DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
//SYSIN    DD  *
 -export EVENTS -port 8778 -userid joe -pwd akkSdiq -level audit
 -stime 2008/04/29,10:00:00 -etime 2008/04/30,10:00:00
 -format ARC -deffile ARCFILE
```

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-export	Output the described broker definition in a format to be used by a broker definition file.
-port	TCP/IP port number of the UEC.
-userid	UEC user ID/account with which Brokers will be modified.
-pwd	Password associated with -userid.
-level	Level of messages written.
-stime	Start time of exported data.
-etime	End time of exported data.
-format	Format of the output from the -export EVENTS action.
-deffile	File containing multiple broker definitions to be added or deleted in the UEC database.

Retrieve Archived File and Export into XML for z/OS

The following figure illustrates the retrieval of an archived file and its export into XML on z/OS.

```
//STEP1    EXEC PGM=UECLOAD,PARM='ENVAR(TZ=EST5EDT)/'
//STEPLIB  DD  DISP=SHR,DSN=#HLQ.UNV.SUNVLOAD
//*
//UNVCONF  DD  DISP=SHR,DSN=#HLQ.UNV.UNVCONF(UECCFG00)
//OUTPUT   DD  SYSOUT=*
//UNVTRACE DD  SYSOUT=*
//ARCFILE  DD  DSN=APP.UEC.ARCH,DISP=SHR
//DEFFILE  DD  DSN=APP.UEC.DEFFILE,DISP=SHR
//SYSOUT   DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
//SYSIN    DD  *
-export EVENTS -arcfile ARCFILE -level audit
-format XML -deffile DEFFILE
```

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-export	Output the described broker definition in a format to be used by a broker definition file.
-arcfile	Archived file to retrieve for export.

-level	Level of messages written.
-format	Format of the output from the -export EVENTS action.
-deffile	File containing multiple broker definitions to be added or deleted in the UEC database.

Maintaining Broker Definitions in UEC Database - Windows

- Export Events into ARC Format for Windows
  - Command Line Options
- Retrieve Archived File and Export into CSV for Windows
  - Command Line Options

Export Events into ARC Format for Windows

The following illustrates the export of events into an ARC format file on Windows.

```
uecload -export EVENTS -userid admin -pwd admin -format ARC -stime 2011/06
/24 -etime 2011/07/24 -deffile c:\test.xml -arcfile c:\test.arc
```

Command Line Options

The command line options used in this example are:

Option	Description
-export	Output the described broker definition in a format to be used by a broker definition file.
-userid	UEC user ID/account with which Brokers will be modified.
-pwd	Password associated with -userid.
-format	Format of the output from the -export EVENTS action.
-stime	Start time of exported data.
-etime	End time of exported data.
-deffile	File containing multiple broker definitions to be added or deleted in the UEC database.
-arcfile	Archived file to retrieve for export.

Retrieve Archived File and Export into CSV for Windows

The following illustrates the retrieval of an archived file and its export into CSV on Windows.

```
uecload -arcfile c:\test.arc -export EVENTS -stime 2011/10/07 -etime 2012/01
/01 -level audit -format CSV -deffile c:\test.csv
```

Note
-port, **-userid**, and **-pwd** are not used, since no connection is made to UEC for this operation.

Command Line Options

The command line options used in this example are:

Option	Description
-arcfile	Archived file to retrieve for export.
-export	Output the described broker definition in a format to be used by a broker definition file.
-stime	Start time of exported data.

-etime	End time of exported data.
-level	Level of messages written.
-format	Format of the output from the -export EVENTS action.
-deffile	File containing multiple broker definitions to be added or deleted in the UEC database.

Event Monitoring and File Triggering

- Introduction
- Detailed Information

Introduction

The Event Monitoring and File Triggering feature of Universal Agent provides a consistent, platform-independent means of monitoring one or more local or remote system events. It also can execute a system command or script based on the outcome of the events that it monitors.

It allows one or more system events to be monitored at any given time.

The methods available for defining an event and its associated actions are described in these pages.

Detailed Information

The following pages provide detailed information for Event Monitoring and File Triggering:

- Event Monitoring and File Triggering - Universal Event Monitor
- Event Monitoring and File Triggering - UEMLoad
- Event Monitoring and File Triggering - Examples

Event Monitoring and File Triggering - Universal Event Monitor

- Universal Event Monitor
  - High-Level Interaction of UEM Components
- Storing Event Definitions and Event Handlers
  - Interaction of Universal Broker and UEM Server during UEMLoad Execution
- Monitoring a Single Event
  - Interaction of Universal Broker and UEM Components during UEM Manager Execution
- Monitoring Multiple Events
  - Interaction of Universal Broker and an Event-Driven UEM Server

Universal Event Monitor

Use the Universal Event Monitor (UEM) Manager to monitor a single local or remote system event.

The UEM Manager (**uem**) may provide all of the parameters necessary to define a system event, or it may specify the ID of a database record that contains the event definition. In either case, the UEM Manager passes the event definition to a local or remote UEM Server (**uemsrv**), which uses that information to look for an occurrence of the event and test for its completion.

The UEM Manager may also provide all of the parameters necessary to define an event handler to the UEM Server, or it may specify the ID of a database record that contains the event handler. An event handler is a command or script that UEM Server executes, based on the outcome of the event occurrence.

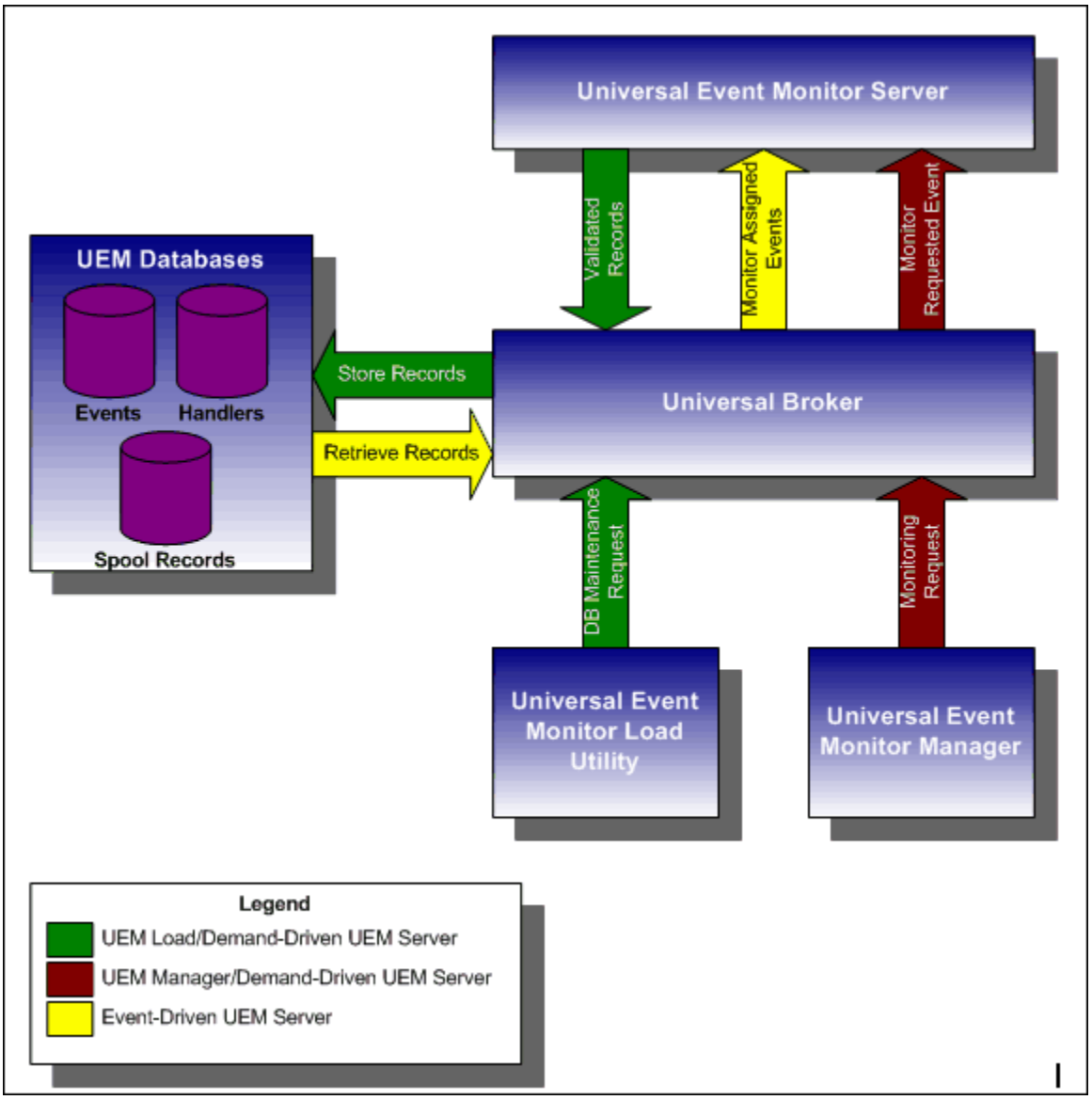
A UEM Server may monitor several local system events simultaneously using records stored in its event definition database. An event-driven UEM Server executes in this manner. An event-driven UEM Server does not require a UEM Manager to initiate a monitoring request, and you may configure it to start automatically whenever the local Universal Broker starts. During start-up, an event-driven UEM Server retrieves a list of its assigned event definitions from the local Universal Broker. UEM Server monitors each event until it is no longer active, or until the event-driven Server ends.

The UEMLoad utility (**uemload**) enables you to add event definition and event handler records to their respective databases

UEMLoad handles all event definition and event handler database management tasks, including adds, updates, deletes, and lists / exports. UEMLoad forwards a database request to a UEM Server, which validates the information. The UEM Server then sends a request to a local Universal Broker to apply the requested operation to the appropriate UEM database file.

High-Level Interaction of UEM Components

The following figure illustrates the interaction of the various components that make up Universal Event Monitor.



Storing Event Definitions and Event Handlers

Event definitions and event handlers can be stored in separate BerkeleyDB database files. When an event definition or event handler record is added to its respective database, a unique identifier must be specified. Whenever UEM is required to monitor an event or execute an event handler, only this ID needs to be referenced in order for UEM to obtain the corresponding event definition or event handler parameters.

UEMLoad initiates all UEM-related database requests. UEMLoad is a command line application that can be used to:

- Add, update, and delete event definition and/or event handlers from their respective databases
- List the entire contents of the event definition and/or event handler databases
- List the parameters of a single event definition and/or event handler
- Export the contents of the event definition and/or event handler databases to a file that can be used to re-initialize the database or populate a new database on another system.

When UEMLoad is started, it sends a request to a Universal Broker running on the local system to start a UEM Server process. Because a client application (that is, UEMLoad) initiates the request, the UEM Server that is started is a demand-driven Server.

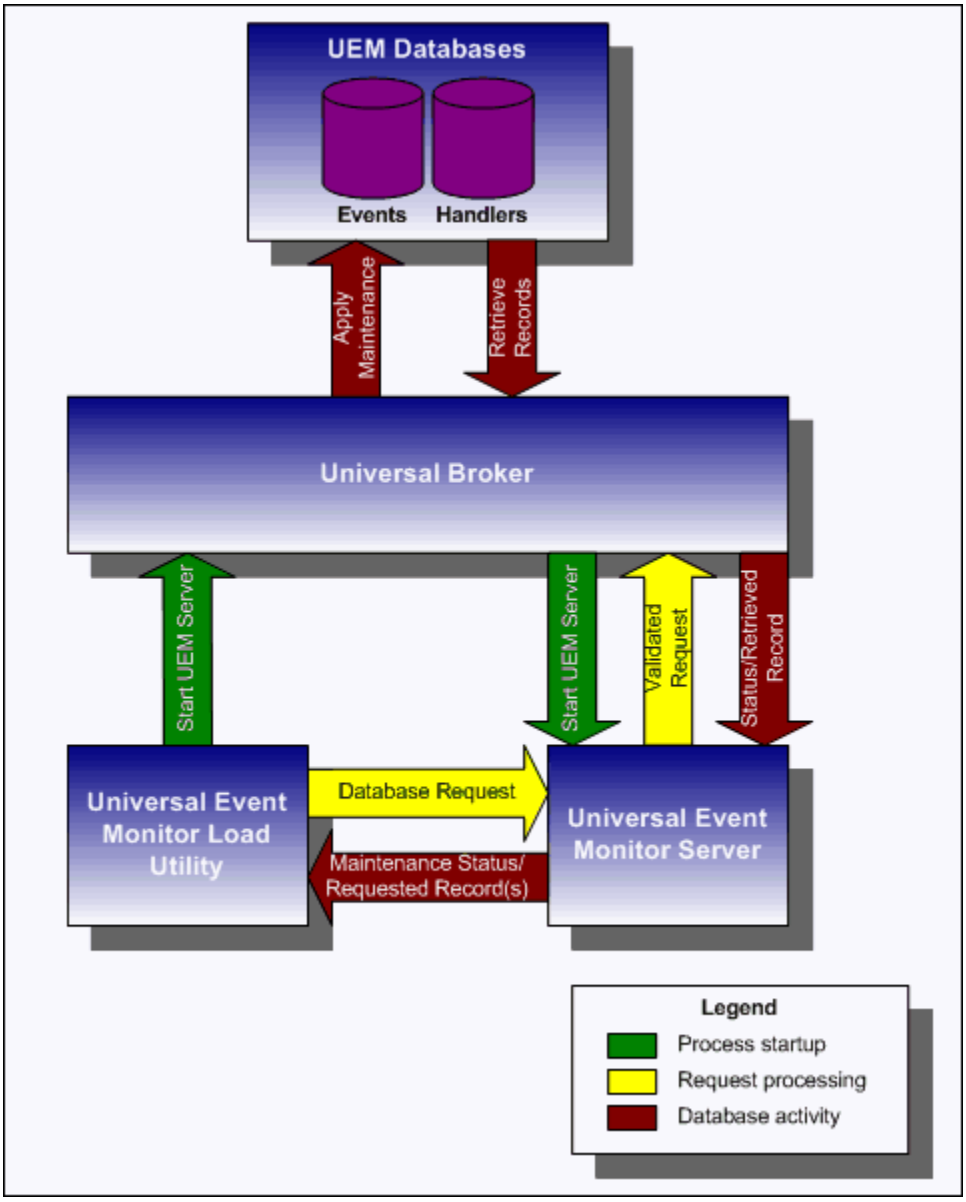
UEMLoad forwards the database request to the UEM Server, which validates it and supplies default values for any required parameters (based upon the type of request) that were not specified from the UEMLoad command line. When a set of complete, valid parameters is available, the UEM Server sends a request to the Universal Broker, which is responsible for actually performing the requested database operation.

Universal Broker reports the success or failure of all database maintenance requests (add, update, delete) to the UEM Server. The UEM Server then passes any errors back to UEMLoad.

For a database query request (list, export), Universal Broker will return the contents of each requested event definition or event handler record to the UEM Server, which then is responsible for forwarding the records to the UEMLoad.

Interaction of Universal Broker and UEM Server during UEMLoad Execution

The following figure illustrates the interaction of the Universal Broker and the Universal Event Monitor Server components involved during the execution of UEMLoad.



Monitoring a Single Event

A single event can be monitored using the UEM Manager. The UEM Manager provides a command line interface from which all parameters required to define an event and its associated event handlers can be specified. In addition, the ID of a stored event definition or event handler can be used as an alternative to specifying all parameters explicitly.

When a UEM Manager is started, it sends a request to the specified local or remote Universal Broker to start a UEM Server. Because the request to start the UEM Server comes from a client application (that is, UEM Manager), it is a *demand-driven* UEM Server that is started.

The UEM Manager sends the monitoring request to the UEM Server. The UEM Server validates the request and supplies default values for any required parameters that were not specified from the command line.

The UEM Manager command line provides for the assignment of an event handler to execute whenever the UEM Server sets the state of an event occurrence or state of the event itself. The UEM Server then is responsible for executing the assigned event handlers which are appropriate for the state change.

The UEM Server will monitor the event until either of the following conditions is satisfied:

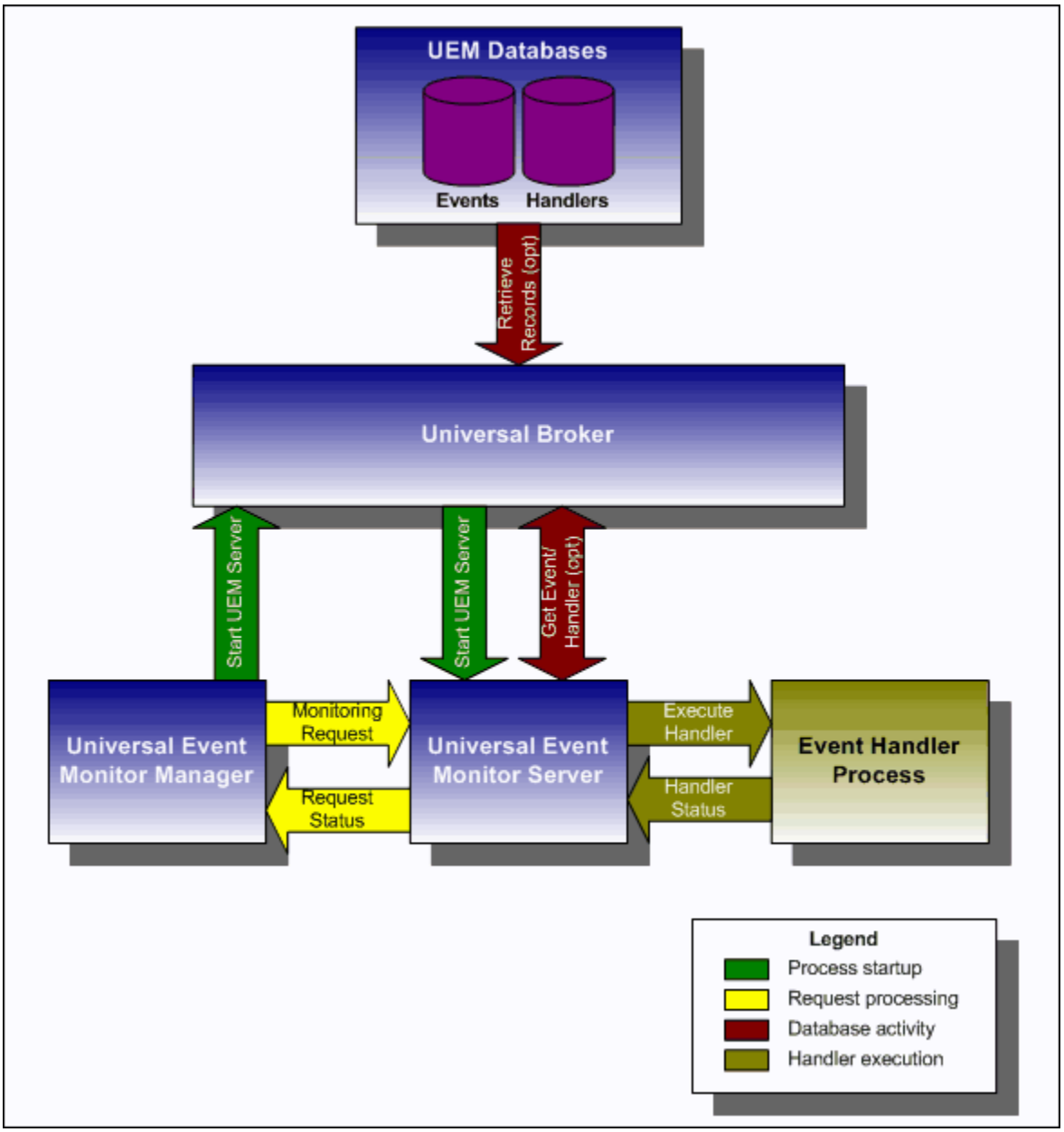
- Required number of expected event occurrences has been detected
- Inactive date and time specified for the event definition elapses.

When either of these occurs, the event becomes inactive and the UEM Server stops monitoring it. The UEM Server then ends after informing the UEM Manager of the result of the monitoring request. The UEM Manager will set its exit code based on this information. This is the default behavior.

However, if an option was set in the UEM Manager instructing it to not wait on the UEM Server, the UEM Manager will end as soon as the UEM Server acknowledges its receipt of a valid monitoring request.

Interaction of Universal Broker and UEM Components during UEM Manager Execution

The following figure illustrates the interaction of the Universal Broker and the Universal Event Monitor components involved when a UEM Manager is executed.



Monitoring Multiple Events

An *event-driven* UEM Server can be used to monitor multiple events at the same time. An event-driven UEM Server uses the records stored in the event definition database file to identify the events it is responsible for monitoring.

An event-driven UEM Server can be executed automatically during start-up of a Universal Broker. While it requires no interaction from a UEM client application, however, an event-driven UEM Server can be started at any time using Universal Control.

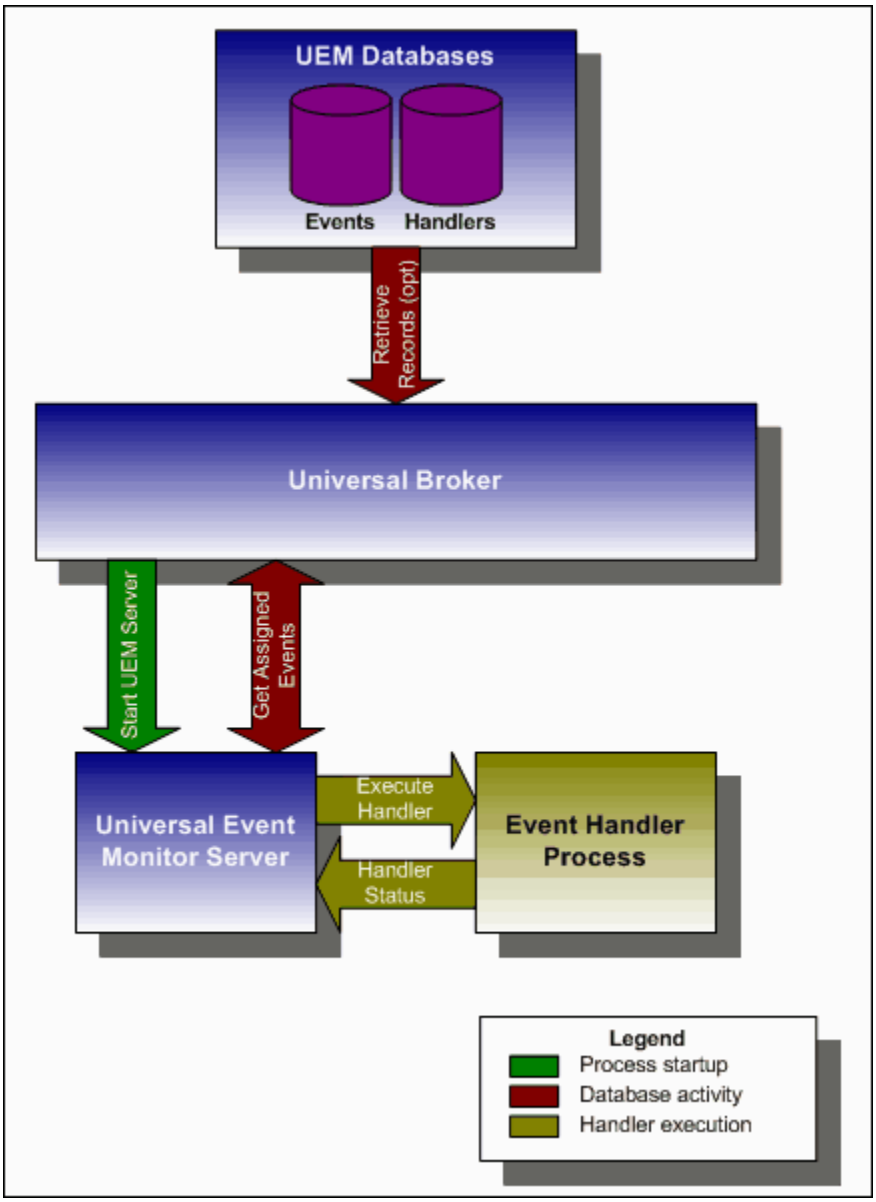
Unless it is stopped manually (using Universal Control), the event-driven UEM Server will continue to run as long as the Broker remains active. When the Broker stops, it will send a stop request to the UEM Server, instructing it to shut itself down.

When an event-driven UEM Server starts, it sends a request to the Broker asking for all of the event definitions residing in the event definition database that are assigned to that event-driven UEM Server. (This assignment was made when the event definition record was added to the database with UEMLoad.) The Server checks the active and inactive dates and times of the event definitions that it receives. It then begins monitoring the active events.

Each event definition provides for the assignment of an event handler to execute when an event occurrence is triggered or rejected. The assignment of an event handler to execute when an event expires also is made within the event definition. The UEM Server is responsible for executing appropriate event handlers based upon the states it sets for detected event occurrences and/or the event themselves.

Interaction of Universal Broker and an Event-Driven UEM Server

The following figure illustrates the interaction of the Universal Broker and an event-driven UEM Server.



- Overview
- Controlling Database Access
  - Access via UEMLoad Utility
  - Universal Access Control List

Overview

A Universal Event Monitor (UEM) Server has three database files that it can use during event processing:

1. **ueme.db** stores event definitions.
2. **uemh.db** stores event handlers.
3. **uems.db** is a spool file that records all activity related to event monitoring.

The UEMLoad utility (**uemload**) manages the event definition and event handler database files. (For information on the spool database file, see Universal Event Monitor Server.)

UEMLoad can be used to:

- Add, update, and delete event definitions and/or event handlers from their respective database files.
- List the entire contents of the event definition and/or event handler database files.
- List the parameters of a single event definition and/or event handler.
- Export the contents of the event definition and/or event handler database files to a file that can be used to re-initialize the database or populate a new database on another system.

By design, UEMLoad itself only can access local event definition and event handler database files. However, it is possible to store definition load files in a single location (for example, a PDS on a z/OS system) and centrally manage their distribution to remote systems using Universal Command.

When a definition load file is redirected from **stdin** to Universal Command, Universal Command will in turn forward the redirected **stdin** to a remote instance of UEMLoad. UEMLoad then behaves as though it were reading a local definition load file.

For detailed information on the event definition and event handler database files, see UEMLoad Utility.

Controlling Database Access

Universal Broker is primarily responsible for providing access to the Universal Agent databases.

However, there are utilities provided, including Universal Spool List (**uslist**) and Universal Spool Remove (**uslrm**) that can be used for direct access to these databases. While these utilities should be used only following a recommendation from and with the assistance of Stonebranch, Inc. Customer Support, they are documented in the Universal Agent Utilities 7.3.x Reference Guide.

To protect the database contents, operating system permissions on the database files themselves should be set so that only accounts with super-user or administrative privileges have access to them.

For more information on the location, names, and contents of the UEM database files, see UEM Server Database Files.

Access via UEMLoad Utility

While the contents of UEM databases can be viewed using Universal Spool List, it is recommended that all access be done using the UEMLoad utility.

The ability to remove event definition and event handler records is provided only with UEMLoad. Universal Spool Remove cannot be used to delete records from those databases.

Only UEMLoad can manage event definition and event handler databases that are local to the system on which the UEMLoad resides. To process a request, the UEMLoad sends a message to the Universal Broker running on that system, instructing it to start a demand-driven UEM Server. A control session is established between UEMLoad and the UEM Server, which provides for direct communication between the two processes.

It is over this session that UEMLoad sends the database request to the UEM Server, so that supplied values can be validated and defaults can be provided for any values that were omitted. The UEM Server then forwards the request to the Universal Broker for actual application of the changes to the appropriate database.

UEMLoad executes in the security context of the user account that started it. Since it is the Universal Broker that applies changes to the event definition and event handler databases, any user with the authority to execute UEMLoad will effectively have access to secure resources. It is therefore strongly recommended that the privileges on UEMLoad be set such that only those user accounts with super-user or administrative privileges be allowed to execute it.

Universal Access Control List

Support for controlling access to the event definition and event handler databases also is provided by UEMLoad.

A type of Universal Access Control List (UACL) is provided in order to grant or deny local user accounts the authority to execute UEMLoad. The type of database access (that is: add, update, delete, list, and export) allowed for each authorized user also can be defined.

A typical set of UACL entries intended to fully secure the event definition and event handler databases would include an entry for each user authorized to execute UEMLoad. Then, the types of database access permitted for each of the users would be set in those entries. Finally, a single UACL entry that denies access to all other accounts would be defined.

Whenever UEMLoad is executed, the entries in the UACL will be checked. If a match cannot be found which indicates that the user account that started UEMLoad has the authority to access the database and perform the requested operation, the application will terminate with an error.

Event Monitoring and File Triggering - Examples

- Introduction
- Event Monitoring and File Triggering Examples - zOS
- Event Monitoring and File Triggering Examples - Windows
- Event Monitoring and File Triggering Examples - UNIX

Introduction

The examples provided here for Event Monitoring and File Triggering are specific to the operating systems supported by Universal Agent.

Links to detailed technical information on appropriate Universal Agent components are provided for each example.

Note
The examples utilizing Universal Event Monitor assume the following information:

- UEM Server is installed on a remote system named **uemhost**.
- Security option has been enabled in the UEM Server's configuration.

The values for the USER_ID (-userid) and USER_PASSWORD (-pwd) configuration options represent the user ID and password of a valid user account defined on **uemhost**.

Event Monitoring and File Triggering Examples - zOS

- Starting an Event-Driven UEM Server - zOS
- Refreshing an Event-Driven UEM Server - zOS
- Using a Stored Event Handler Record - z/OS
- Handling an Event with a Script - z/OS
- Handling an Expired Event - z/OS
- Continuation Character (-) in z/OS Handler Script
- Continuation Character (+) in z/OS Handler Script
- Continuation Characters (- and +) in z/OS Handler Script

Event Monitoring and File Triggering Examples - Windows

- Using a Stored Event Handler Record - Windows
- Execute Script for Triggered Event Occurrence - Windows
- Handling an Expired Event - Windows
- Add a Single Event Record - Windows
- Add a Single Event Handler Record - Windows
- List All Event Definitions - Windows
- Export Event Definition and Handler Databases - Windows
- List a Single Event Handler Record - Windows
- List Event Definitions and Handlers Using Wildcards - Windows
- Add Record(s) Using Definition File - Windows
- Add Records Remotely Redirected from STDIN - Windows
- Add Records Redirected from STDIN (for z/OS) - Windows
- Definition File Format - Windows

Event Monitoring and File Triggering Examples - UNIX

- Using a Stored Event Handler Record - UNIX
- Execute Script for Triggered Event Occurrence - UNIX

- Handling an Expired Event - UNIX
- Add a Single Event Record - UNIX
- Add a Single Event Handler Record - UNIX
- List All Event Definitions - UNIX
- Export Event Definition and Handler Databases - UNIX
- List a Single Event Handler Record - UNIX
- List Event Definitions and Handlers Using Wildcards - UNIX
- Add Record(s) Using Definition File - UNIX
- Add Record(s) Remotely Redirected from STDIN - UNIX
- Add Record(s) Remotely Redirected from STDIN (for z/OS) - UNIX
- Definition File Format - UNIX

Starting an Event-Driven UEM Server - zOS

Starting an Event-Driven UEM Server

There are two ways start an event-driven UEM Server (**uems**) component:

1. Recycle the **ubroker** daemon (Universal Broker service under Windows).
2. Use Universal Control to start the **uems**, either locally on the server or from the mainframe.

In this example, **uems** is started from the mainframe.

(This job will fail if **uems** is running at the time of submit; **uems** usually is started by the Universal Broker when it is started.)

```
//STUEMS   JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//         JCLLIB ORDER=SBI.UNV.SUNVSAMP
//*
//STEP1    EXEC UCTLPRC
//LOGONDD  DD  DISP=SHR,DSN=MFC1A.JCL.CNTRL(WINUSER)
//SYSIN    DD  *
-host 172.16.30.30 -encryptedfile LOGONDD -port 7887 -start uems
/*
```

Note
There is only one different command (**-start**) between this example and Refreshing an Event-Driven UEM Server.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-host	TCP/IP host name of the remote computer on which Universal Broker is running and accepting connections.
-encrypted	Encrypted command file.
-port	TCP/IP port number of the remote computer on which Universal Broker is running and accepting connections.
-start	Instruction to a Universal Broker to start the UEM Server.

Components

Universal Control

Universal Event Monitor Server for Windows

Universal Event Monitor Server for UNIX

Refreshing an Event-Driven UEM Server - zOS

Refreshing an Event-Driven UEM Server

In this example, RESUEMS will refresh the event-driven UEM Server (**uems**) to secure changes made to the configuration file.

```
//RESUEMS  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//         JCLLIB ORDER=SBI.UNV.SUNVSAMP
//*
//STEP1    EXEC UCTLPRC
//LOGONDD  DD  DISP=SHR,DSN=MFC1A.JCL.CNTRL(WINUSER)
//SYSIN    DD  *
-host 172.16.30.30 -encryptedfile LOGONDD -port 7887 -refresh uems
/*
```

Note
There is only one different command (**-refresh**) between this example and Starting an Event-Driven UEM Server.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-host	TCP/IP host name of the remote computer on which Universal Broker is running and accepting connections.
-encrypted	Encrypted command file.
-port	TCP/IP port number of the remote computer on which Universal Broker is running and accepting connections.
-refresh	Instruction to the Universal Broker to refresh the UEM Server configuration.

Components

Universal Control

Universal Event Monitor Server for Windows

Universal Event Monitor Server for UNIX

Using a Stored Event Handler Record - zOS

Using a Stored Event Handler Record in z/OS

In this example, a demand-driven UEM Server will watch for the creation of a file called **uemtest.dat**. Since no path is specified, it will look for this file in the user's UEM Server working directory, as specified in the component definition for a demand-driven UEM Server.

If the file completes before the inactive time of *17:38* elapses, the event occurrence will be set to the **triggered** state, and UEM will execute the command or script contained in the event handler **h001**, which is the ID of a record in the event handler database.

If the file does not complete before the inactive time elapses, the event occurrence will be set to a **rejected** state. Since no event handler information is provided for a rejected occurrence, no further action will be taken by the UEM Server.

If the UEM Server does not detect the presence of **uemtest.dat** before the inactive time of *17:38* elapses, the event will be set to an **expired** state.

Note
Because the inactive date value was omitted, UEM Manager will default the inactive date to the current date. Further, because no handler information is given for the **expired** state, no further action will be taken by the UEM Server once the event expires.

```
//jobname  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//STEP1    EXEC UEMPRC
//SYSIN    DD *
-event_type        file
-filespec          uemtest.dat
```

```
-wait               yes
-inact_date_time    ,17:38
-triggered
-handler_id         h001
-host               uemhost
-userid             uemuser
-pwd                uemusers_password
-max_count          1
/*
```

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-event_type	Type of event to monitor.
-filespec	Name or pattern of the file whose creation should be detected and tracked for completion.
-wait	Forces the UEM Manager to wait for the completion of the UEM Server.
-inact_date_time	Date and time at which the state of the monitored event should be made inactive.
-triggered	Event state that, when encountered, will result in the execution of the associated event handler.
-handler_id	ID of a stored event handler record.
-host	List of one or more hosts upon which a command may run.
-userid	ID of a remote user account that the UEM Server uses to establish the security context in which event monitoring is performed.
-pwd	Password associated with -userid.
-max_count	Maximum number of event occurrences to monitor.

Components

Universal Event Monitor Manager for z/OS

Universal Event Monitor Server

Handling an Event with a Script - zOS

Handling an Event With a Script in z/OS

In this example, a demand-driven UEM Server installed on a Windows machine will watch for the creation of a file called **uemtest.dat**. Since no path is specified, it will look for this file in the user's UEM Server working directory.

A relative inactive date/time is used to instruct the UEM Server to monitor the event for *10* minutes. If the file is detected and completes within that time, the event occurrence will be set to the **triggered** state. The script statements contained within the **MYSCRIPT** DD statement then will be written to a temporary script file and executed by UEM Server.

The value specified by the **-handler_opts** option is appended to the command line constructed by UEM in order to execute the temporary script file. This will cause the values *parm1*, *parm2*, and *parm3* to be passed to the script. Further, any output generated by the script will be written to a file in the UEM Server working directory, **uemtest.log**.

If the file is detected, but does not complete before the inactive time elapses, the event occurrence will be set to a **rejected** state. Since no event handler information is provided for a rejected occurrence, no further action will be taken by the UEM Server.

If the UEM Server does not detect the presence of **uemtest.dat** before the inactive time elapses, the event will be set to an **expired** state. Again, because no handler information is given for this state, no further action will be taken by the UEM Server.

```
//jobname  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//STEP1    EXEC UEMPRC
```

```
//MYSCRIPT DD *
@echo off

:: Program variables
set parmCtr=1

:: Loop through parameter list

:: **** Start of loop ****
:BeginLoop
if ""%1""==""""" goto EndLoop

:DisplayParm
echo Parm %parmCtr%: %1

:: Shift the next parm
shift
set /a parmCtr+=1

:: Go back to the top
goto BeginLoop

:: **** End of loop ****
:EndLoop

//SYSIN    DD *
 -event_type        file
 -filespec          uemtest.dat
 -inact_date_time   +10
 -handler_opts      "parm1 parm2 parm3 >uemtest.log 2>&1"
 -host              uemhost
 -userid            uemuser
 -pwd               uemusers_password
 -triggered -script myscript
/*
```

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-event_type	Type of event to monitor.
-filespec	Name or pattern of the file whose creation should be detected and tracked for completion.
-inact_date_time	Date and time at which the state of the monitored event should be made inactive.
-handler_opts	Forces the UEM Manager to wait for the completion of the UEM Server.
-host	List of one or more hosts upon which a command may run.
-userid	ID of a remote user account that the UEM Server uses to establish the security context in which event monitoring is performed.
-pwd	Password associated with -userid.
-triggered	Event state that, when encountered, will result in the execution of the associated event handler.
-script	

	Complete path to a local script file or DD statement that contains one or more system commands that should be executed on behalf of the event handler.

Components

Universal Event Monitor Manager for z/OS

Universal Event Monitor Server for Windows

Handling an Expired Event - zOS

Handling an Expired Event in z/OS

In this example, a demand-driven UEM Server installed on a UNIX system watches for the creation of a file called **uemtest.dat**. The **-filespec** option contains no path information, so UEM Server looks for this file in uemuser's home directory.

A relative inactive date / time instructs the demand-driven Server to monitor the event for one (1) minute. If the UEM Server detects the file, and the file completes within that time, UEM sets the event occurrence to the **triggered** state. Since the command options contain no event handler information for a **triggered** occurrence, the UEM Server simply renames the file (by default). UEM Server then makes the event inactive, and ends.

If the UEM Server detects the file, but the file does not complete within 60 seconds, UEM sets the occurrence to the rejected state. Since the command options contain no event handler information for a rejected occurrence, the UEM Server leaves the file as-is and takes no further action.

If the UEM Server does not detect the presence of uemtest.dat before the inactive time elapses, it sets the event to the expired state. When this happens, the UEM Server executes the command specified by the **-cmd** parameter of the **-expired** option. In this example, UEM executes the **ls -alR /home** command.

Note
In this example, the -expired option is followed by the -options option, which redirects the output of the **"ls -alR /home"** command to a file in uemuser's home directory called **uemtest.log**.

```
//jobname  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//STEP1    EXEC UEMPRC
//SYSIN    DD *
-event_type        file
-filespec          uemtest.dat
-inact_date_time   +1
-expired -cmd "ls -alR /home" -options ">uemtest.log 2>&1"
-host              uemhost
-userid            uemuser
-pwd               uemusers_password
/*
```

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-event_type	Type of event to monitor.
-filespec	Name or pattern of the file whose creation should be detected and tracked for completion.
-inact_date_time	Date and time at which the state of the monitored event should be made inactive.
-expired	Event state that, when encountered, will result in the execution of the associated event handler.
-cmd	Complete path to an application file or remote script that should be executed on behalf of the event handler.
-host	List of one or more hosts upon which a command may run.
-userid	

	ID of a remote user account that the UEM Server uses to establish the security context in which event monitoring is performed.
-pwd	Password associated with -userid.

Components

Universal Event Monitor Manager for z/OS

Universal Event Monitor Server for UNIX

Continuation Character (-) in zOS Handler Script

Continuation Character - in z/OS Handler Script

Continuation characters (**-** and **+**) are useful when you want to execute a script line that is longer than your available z/OS character space.

The **-** continuation character will preserve trailing spaces in your line.

The **+** continuation character will not preserve trailing spaces in your line.

```
The following z/OS handler script:

begin_script
 stmt "ls -a -             <---- Notice the continuation character "-"
 >dirfile"
 end_script

Will produce the following output when loaded to the uemh.db:

Handler ID...................: MFCTRIGGER_1
Handler Type.................: SCRIPT
Max Acceptable Return Code...: 0
Encrypted User File..........:
User ID......................: mfc1a
Script statements:
 ls -a  >dirfile
 Script Type..................: bat
Command Line Options.........:
Last Modified On.............: 06/11/11 10:32:31 AM
Last Modified By.............: mfc1a
```

Components

Universal Event Monitor Manager for z/OS

Continuation Character (+) in zOS Handler Script

Continuation Character + in z/OS Handler Script

Continuation characters (**-** and **+**) are useful when you want to execute a script line that is longer than your available z/OS character space.

The **-** continuation character will preserve trailing spaces in your line.

The **+** continuation character will not preserve trailing spaces in your line.

```
The following z/OS handler script:

begin_script
 stmt "ls -a >dir +          <---- Notice the continuation character "+"
```

```
 file"
 end_script

Will produce the following output when loaded to the uemh.db:

Handler ID...................: MFCTRIGGER_1
Handler Type.................: SCRIPT
Max Acceptable Return Code...: 0
Encrypted User File..........:
User ID......................: mfc1a
Script statements:
 ls -a >dirfile
Script Type..................: bat
Command Line Options.........:
Last Modified On.............: 06/11/11 11:46:32 AM
Last Modified By.............: mfc1a
```

Components

Universal Event Monitor Manager for z/OS

Continuation Characters (- and +) in zOS Handler Script

Continuation Characters - and + in z/OS Handler Script

Continuation characters (**-** and **+**) are useful when you want to execute a script line that is longer than your available z/OS character space. The **-** character will preserve trailing spaces in your line. The **+** character will not preserve trailing spaces in your line.

This example shows the use of **+** to concatenate a command line or a word within a z/OS script without a space as the use of **-** to continue a line of script where a space is required within the same z/OS handler script.

```
The following z/OS handler script:

 begin_script
 stmt "ls -a >dir +
 file"
 stmt "uemFName=`basename \u201c$UEMORIGFILE\u201d | sed 's/\(.*\)+
 \.\(.*$\)/\1/'`"
 stmt "fname=$uemFName.$dt.$tm.$pid.txt"
 stmt " ls -al >dir+
 data"
 stmt "ls -a -
 >new+
data"
 end_script

Will produce the following output when loaded to the uemh.db:

Handler ID...................: MFCTRIGGER_1
Handler Type.................: SCRIPT
Max Acceptable Return Code...: 0
Encrypted User File..........:
User ID......................: mfc1a
Script statements:
 ls -a >dirfile
 uemFName=`basename \u201c$UEMORIGFILE\u201d | sed 's/\(.*\)\.\(.*\)/\1/'`
 fname=$uemFName.$dt.$tm.$pid.txt
```

```
  ls -al >dirdata
  ls -a  >newdata
Script Type..................: bat
Command Line Options.........:
Last Modified On.............: 06/11/1 01:25:20 PM
Last Modified By.............: mfc1a
```

Components

Universal Event Monitor Manager for z/OS

Using a Stored Event Handler Record - Windows

Using a Stored Event Handler Record in Windows

In this example, a demand-driven UEM Server will watch for the creation of a file called **uemtest.dat** in the **`C:\UEM Files`** directory.

If the file completes before the inactive time of *20:00* elapses, the event occurrence will be set to the **triggered** state, and UEM will execute the command or script contained in the event handler **h001**, which is the ID of a record in the event handler database.

If the file does not complete before the inactive time elapses, the event occurrence will be set to a rejected state. Since no event handler information is provided for a rejected occurrence, no further action will be taken by the UEM Server.

If the UEM Server does not detect the presence of **`C:\UEM Files\uemtest.dat`** before the inactive time of *20:00* elapses, the event will be set to an **expired** state.

> **Note**
> Because the inactive date value was omitted, UEM Manager will default the inactive date to the current date. Further, because no handler information is given for the **expired** state, no further action will be taken by the UEM Server once the event expires.

```
uem -host uemhost -event_type file
-filespec "C:\UEM Files\uemtest.dat"
-inact_date_time ,20:00 -userid uemuser -pwd uemusers_password
-triggered -handler_id h001
```

Command Line Options

The command line options used in this example are:

Option	Description
-host	List of one or more hosts upon which a command may run.
-event_type	Type of event to monitor.
-filespec	Name or pattern of the file whose creation should be detected and tracked for completion.
-inact_date_time	Date and time at which the state of the monitored event should be made inactive.
-userid	ID of a remote user account that the UEM Server uses to establish the security context in which event monitoring is performed.
-pwd	Password associated with -userid.
-triggered	Event state that, when encountered, will result in the execution of the associated event handler.
-handler_id	ID of a stored event handler record.

Components

Universal Event Monitor Manager for Windows

Execute Script for Triggered Event Occurrence - Windows

- Executing a Script for a Triggered Event Occurrence in Windows
  - Command Line Options
  - Contents of Sample Script File
  - Components

Executing a Script for a Triggered Event Occurrence in Windows

In this example, a demand-driven UEM Server installed on a UNIX machine will watch for the creation of a file called **uemtest.dat**. Since no path is specified, it will look for this file in the user's home directory.

A relative inactive date / time is used to instruct the UEM Server to monitor the event for *10* minutes. If the file is detected and completes within that time, the event occurrence will be set to the **triggered** state. The script statements contained within the local file **`C:\UEMScripts\h_001.txt`** then will be written to a temporary script file on **uemhost** and executed by UEM Server.

The value specified by the **-handler_opts** option is appended to the command line constructed by UEM to execute the temporary script file. This will cause the values *parm1*, *parm2*, and *parm3* to be passed to the script. Any output generated by the script will be written to a file in the UEM Server working directory called **uemtest.log**.

If the file is detected, but does not complete before the inactive time elapses, the event occurrence will be set to a **rejected** state. Since no event handler information is provided for a rejected occurrence, no further action will be taken by the UEM Server.

If the UEM Server does not detect the presence of **uemtest.dat** before the inactive time elapses, the event will be set to an **expired** state. Again, because no handler information is given for this state, no further action will be taken by the UEM Server.

```
uem -host uemhost -event_type file -filespec uemtest.dat
-inact_date_time +10 -userid uemuser -pwd uemusers_password
-triggered -script C:\UEMScripts\h_001.txt
-handler_opts "parm1 parm2 parm3 >uemtest.log 2>&1"
```

Command Line Options

The command line options used in this example are:

Option	Description
-host	List of one or more hosts upon which a command may run.
-event_type	Type of event to monitor.
-filespec	Name or pattern of the file whose creation should be detected and tracked for completion.
-inact_date_time	Date and time at which the state of the monitored event should be made inactive.
-userid	ID of a remote user account that the UEM Server uses to establish the security context in which event monitoring is performed.
-pwd	Password associated with -userid.
-triggered	Event state that, when encountered, will result in the execution of the associated event handler.
-script	Complete path to a local script file that contains one or more system commands that should be executed on behalf of the event handler.
-handler_opts	Options that are passed as command line arguments to any process executed on behalf of an event handler.

Contents of Sample Script File

The following figure illustrates the contents of the **`C:\UEMScripts\h_001.txt`** file.

```
#!/bin/sh

# Sample script h_001.txt

argNum=1
```

```
# Display each command line argument.
while [ "$1" != "" ]
do
echo Parm $argNum: $1
shift
argNum=`expr $argNum + 1`
done
```

Components

Universal Event Monitor Manager for Windows

Universal Event Monitor Server for UNIX

Handling an Expired Event - Windows

Handling an Expired Event in Windows

In this example, a demand-driven UEM Server installed on a UNIX system watches for the creation of a file called **uemtest.dat** in the **/uem** files directory.

> **Note**
> The space that precedes the path name specified in the **-filespec** option is necessary to accommodate parsing requirements for command options in Windows (see the UEM Manager FILE_SPECIFICATION option).

A relative inactive date / time instructs the demand-driven Server to monitor the event for one (1) minute. If the UEM Server detects the file, and the file completes within that time, UEM sets the event occurrence to the triggered state. Since the command options contain no event handler information for a triggered occurrence, the UEM Server simply renames the file (by default). UEM Server then makes the event inactive, and ends.

If the UEM Server detects the file, but the file does not complete within 60 seconds, UEM sets the occurrence to the rejected state. Since the command options contain no event handler information for a rejected occurrence, the UEM Server leaves the file as-is and takes no further action.

If the UEM Server does not detect the presence of uemtest.dat before the inactive time elapses, it sets the event to the expired state. When this happens, the UEM Server executes the command specified by the **-cmd** parameter of the **-expired** option. In this example, UEM executes the **'ls -alR /uem files'** command.

> **Note**
> In this example, the -expired option is followed by the -options option, which redirects the output of the **"ls -alR '/uemfiles'"** command to a file in uemuser's home directory called **uemtest.log**.

```
uem -host uemhost -event_type file
-userid uemuser -pwd uemusers_password
-filespec " /uem files/uemtest.dat"
-inact_date_time +1
-expired -cmd "ls -alR '/uem files'" -options ">uemtest.log 2>&1"
```

Command Line Options

The command line options used in this example are:

Option	Description
-host	List of one or more hosts upon which a command may run.
-event_type	Type of event to monitor.
-userid	ID of a remote user account that the UEM Server uses to establish the security context in which event monitoring is performed.

-pwd	Password associated with -userid.
-filespec	Name or pattern of the file whose creation should be detected and tracked for completion.
-inact_date_time	Date and time at which the state of the monitored event should be made inactive.
-expired	Event state that, when encountered, will result in the execution of the associated event handler.
-cmd	Complete path to an application file or remote script that should be executed on behalf of the event handler.
-options	Values that are passed as command line arguments to a particular handler specified for a given EVENT_STATE.

Components

Universal Event Monitor Manager for Windows

Universal Event Monitor Server for UNIX

Add a Single Event Record - Windows

Adding a Single Event Record for Windows

In this example, a single event record identified as **payrollfile** is added to the local event definition database.

This event definition will instruct a UEM Server, which resides on the local (UNIX) system, to detect all occurrences of the file **/tmp/payroll.dly**. Whenever UEM detects this file and sets the associated event occurrence to a **triggered** state, UEM will execute the command or script contained in the stored event handler record that has an ID of **listdir**. If this event handler record does not exist at the time the event occurrence is triggered, an error will be issued by UEM.

When the record has been added to the event definition database, it is immediately available for use by a demand-driven UEM Server. In other words, there is no restriction with respect to how quickly a UEM Manager can reference the stored event definition after UEMLoad adds it to the database.

Because no values for the UEMLoad EVENT_STATE, ACTIVE_DATE_TIME, and INACTIVE_DATE_TIME options were specified, the default value of **enable**, the current date and time, and 2038.01.16,23:59, respectively, are used. This means the event will be monitored as soon as the event definition is assigned to an event-driven UEM Server. In this case, the event definition is assigned to the UEM Server component with an ID of **uems** (the default).

If this UEM Server component is active when the record is added, this assignment will occur the next time that the UEM Server refreshes its configuration. If the UEM Server component is not active, the assignment is made the next time it is started.

```
uemload -add -event_id payrollfile -event_type file
-filespec "/tmp/payroll.dly" -triggered_id listdir
```

Command Line Options

The command line options used in this example are:

Option	Description
-add	Writes one or more new event definition and/or event handler records to the appropriate database.
-event_id	Identifier that uniquely identifies an event definition record.
-event_type	Type of system event represented by the event definition record.
-filespec	Name of a file to monitor.
-triggered_id	ID of an event handler record that UEM will execute when an event occurrence is triggered.

Components

UEMLoad Utility for Windows

Universal Event Monitor Server for UNIX

Add a Single Event Handler Record - Windows

Adding a Single Event Handler Record for Windows

In this example, a single handler record identified, **listdir**, is added to the local event handler database.

Whenever a UEM Server invokes this event handler, a handler process is started that executes the command **ls -al**, which lists the contents of the current directory on a UNIX system. The **encrypted.file** file, referenced by the **-encryptedfile** option, is a Universal Encrypted file. This file contains a user ID and, optionally, a password that is used by an event-driven UEM Server to establish a security context in which to execute the handler process (provided the USER_SECURITY option is enabled in the UEM Server's configuration).

Once this record is added, it is available immediately to both demand-driven and event-driven UEM Servers.

Note
If a demand-driven UEM Server uses this handler, any user information specified in **encrypted.file** is overridden by the user information provided by the UEM Manager's command options.

```
uemload -add -handler_id listdir -encryptedfile encrypted.file
-cmd "ls -al"
```

Command Line Options

The command line options used in this example are:

Option	Description
-add	Writes one or more new event definition and/or event handler records to the appropriate database.
-handler_id	Identifier that uniquely identifies an event handler record.
-encryptedfile	Complete path to a file encrypted with Universal Encrypt.
-cmd	Command to execute on behalf of the event handler.

Components

UEMLoad Utility for Windows

Universal Event Monitor Server for UNIX

Universal Encrypt

List All Event Definitions - Windows

Listing All Event Definitions for Windows

In this Windows example, the **-list** option is used to dump all records in the event definition database and display them to **stdout**.

If the request were executed on a UNIX system, the asterisk (*****) would need to be escaped or enclosed within quotes (that is: ***** or **"*"**, respectively).

```
uemload -list -event_id *
```

Note
The default behavior when listing or exporting records, when neither an event ID nor a handler ID is specified, is to return all records. However, in this example, even though no handler ID was specified, no event handler records are returned.

Conversely, if just a handler ID had been specified, no event definition records would be returned. Supplying an event ID and/or handler ID serves as a filter which causes **uemload** to return just those records specifically requested.

Command Line Options

The command line options used in this example are:

Option	Description
-list	Displays the complete contents of the specified event definition and/or event handler records.
-event_id	Identifier that uniquely identifies an event definition record.

Components

UEMLoad Utility for Windows

Export Event Definition and Handler Databases - Windows

Exporting the Event Definition and Event Handler Databases for Windows

In this example, the -export option is used to dump all records in the event definition and event handler databases to a text file in the current directory named **uemout.txt**. This file is a UEMLoad definition file that also can be used to add or update records in the event definition and/or event handler databases.

The contents of the file resembles the example shown in Definition File Format - Windows.

```
-export -deffile uemout.txt
```

Note
No event ID or handler ID is specified from the command line. If neither parameter is specified when listing or exporting records, the default behavior is to retrieve all database records.

Command Line Options

The command line options used in this example are:

Option	Description
-export	Dumps the contents of the specified event definition and/or event handler records to a text file that can be used as input to a subsequent run of the UEMLoad utility.
-deffile	Name of a file that contains event definition and/or event handler parameters.

Components

UEMLoad Utility for Windows

List a Single Event Handler Record - Windows

List a Single Event Handler Record for Windows

In this example, the **-list** option is used to display the contents of an event handler record with an ID of **dirlist**.

```
uemload -list -handler_id dirlist
```

The following figure illustrates sample output for this command. (The values shown are those that could be expected if the record were added using the command shown in Add a Single Event Handler Record - Windows.)

In this specific instance, the user ID contained in **encrypted.file** (from Add a Single Event Handler Record - Windows) is **sparkie**, and the record was added by the user account with an ID of **sbuser**.

Command Line Options

The command line options used in this example are:

Option	Description
-list	Displays the complete contents of the specified event definition and/or event handler records.
-handler_id	Identifier that uniquely identifies an event handler record.

Sample List Output

```
UNV3659I Connecting to local broker on port 7887.
UNV3406I Universal Event Monitor Server component 1117035117 started.
UNV3666I Load request started at 11:32:45 AM 05/25/2011.

Event Handler(s):
=================
Handler ID.................: dirlist
Max Acceptable Return Code.: 0
User ID....................: sparkie
Command....................: ls -al
Last Modified On...........: 05/25/2011 11:32:06 AM
Last Modified By...........: sbuser

UNV3667I Universal Event Monitor Load is ending successfully with exit code
0.
```

Components

UEMLoad Utility for Windows

List Event Definitions and Handlers Using Wildcards - Windows

Listing Multiple Event Definitions and Event Handlers Using Wildcards for Windows

In this example, the wildcards supported by **uemload** are demonstrated.

Wildcards can be used to select event definitions and event handlers whose respective IDs match the specified pattern.

- Asterisk (*****) can be used to match 0 or more characters.
- Question mark (**?**) can be used to match any single character.

All event definitions whose IDs start with the characters **event** are returned by the command below. In addition, all event handlers whose IDs begin with **handler0** and end with any two characters are selected.

```
uemload -list -event_id event* -handler_id handler0??
```

Command Line Options

The command line options used in this example are:

Option	Description
-list	Displays the complete contents of the specified event definition and/or event handler records.
-event_id	Identifier that uniquely identifies an event definition record.
-handler_id	Identifier that uniquely identifies an event handler record.

Components

UEMLoad Utility for Windows

Add Record(s) Using Definition File - Windows

Add Record(s) Using a Definition File for Windows

In this example, a text file named **uemadd.txt** is used to add one or more records to the UEM databases. The contents of the file resemble those shown in Definition File Format - Windows.

A definition file allows multiple records to be added to the event definition and/or event handler databases at the same time. When no definition file is used, only a single record can be added to the database(s).

```
uemload -add -deffile uemadd.txt
```

Command Line Options

The command line options used in this example are:

Option	Description
-add	Writes one or more new event definition and/or event handler records to the appropriate database.
-deffile	Name of a file that contains event definition and/or event handler parameters.

Components

UEMLoad Utility for Windows

Add Records Remotely Redirected from STDIN - Windows

Add Record(s) Remotely, Using a Definition File Redirected from STDIN for Windows

In this example, a definition load file named **uemadd.txt** is used to add one or more records to the databases of a remote UEM Server. The contents of the file resemble those shown in Definition File Format - Windows.

Universal Command is used to execute UEMLoad on the remote UEM Server's system. The definition load file is redirected from standard input (**stdin**), which eliminates the step of copying the load file to the remote system before executing UEMLoad.

```
ucmd -cmd "uemload -add" -host rmthost -encryptedfile rmtacctinfo.enc
<uemadd.txt
```

Command Line Options

The command line options used in this example are:

Option	Description
-cmd	Remote command to execute.
-add	Writes one or more new event definition and/or event handler records to the appropriate database.
-host	List of one or more hosts upon which a command may run.
-encryptedfile	Encrypted command file.

Components

UEMLoad Utility for Windows

Universal Command Manager for Windows

Universal Event Monitor Server

Add Records Redirected from STDIN (for zOS) - Windows

Add Record(s) Remotely, Using a Definition File Redirected from STDIN (for z/OS) for Windows

In this example, a definition load file named **MY.UEM.DATA(UEMDEF)** is used to add one or more records to the databases of a remote UEM Server. The contents of the file resemble those shown in Definition File Format - Windows.

Universal Command is used to execute UEMLoad on the remote UEM Server's system. It redirects standard input (stdin) from a data set allocated to the UNVIN ddname. This eliminates the step of copying the data set to the remote system before executing UEMLoad.

```
//STEP1    EXEC UCMDPRC
//UNVIN    DD  DISP=SHR,DSN=MY.UEM.DATA(UEMDEF)
//SYSIN    DD  *
-host                dallas
-userid              joe
-pwd                 ahzidaeh
-cmd                 "uemload -add"
```

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-host	List of one or more hosts upon which a command may run.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-cmd	Remote command to execute.
-add	Writes one or more new event definition and/or event handler records to the appropriate database.

Components

UEMLoad Utility for Windows

Universal Command Manager for z/OS

Universal Event Monitor Server

Definition File Format - Windows

Definition File Format for Windows

The format of the definition load file for events and event handlers follows the keyword / value-pair convention used for Universal Agent configuration files. However, because more than one definition can be specified in a load file, some additional conventions are used.

- The **begin_event** and **end_event** keywords are used to mark the beginning and end, respectively, of each event definition entry.
- The **begin_handler** and **end_handler** keywords are used to mark the beginning and end, respectively, of each event handler entry.
- The **begin_script** and **end_script** keywords are used to mark the beginning and end, respectively, of any user script contained in the definition load file.

Lines that belong to the script must begin with the **stmt** keyword. Long **stmt** values that have to be split across lines can be done so using the **+** and **-** line continuation characters (as described in Configuration File Syntax). These lines will be accepted verbatim, and no script syntax validation will be done. Lines will continue to be added to the script until an **end_script**, **end_handler**, **begin_handler**, or **begin_event** keyword is read, or the end of the file is reached.

If a parameter's value contains spaces, it must be enclosed in double (**"**) quotation marks.

If quotes are to be saved as part of the parameter's value, use extra double (**"**) quotation marks to escape the quotes (for example, **optname "optval1 ""optval2 optval2a"" optval3"**).

The **script** keyword can be used in lieu of a **begin_script/end_script** block, in which case the contents of the specified file will be written to the event handler.

A sample definition file for Windows is shown in the following figure.

```
# Indented lines are for illustration only. Leading spaces are
# ignored by UEMLoad. Defaults will be used for any omitted
# values.

# Start of parameters for an event definition with an ID of
# "win_event_sample".

begin_event
   event_id win_event_sample
   event_type FILE
   comp_name uems
   state enable
   inact_date_time 2011.12.31,23:59
   triggered_id script_sample
   filespec "uem*.dat"
   rename_file yes
   rename_filespec "$(compname).$(compid).$(date).$(seqno)"
end_event

# End of parameters for event definition "win_event_sample".

# Start of parameters for an event handler with an ID of
# "win_script_sample".

begin_handler
   handler_id script_sample
   handler_type SCRIPT
   maxrc 0
   userid uemuser
   begin_script
      stmt "@echo off"
      stmt ""
      stmt "dir /-p/o/s ""C:\Program Files"""
   end_script
   script_type bat
end_handler

# End of parameters for event handler "win_script_sample".

# Start of parameters for an event definition with an ID of
# "win_cmd_sample".

begin_handler
   handler_id cmd_sample
   maxrc 0
   userid uemuser
   cmd "C:\Documents and Settings\uemuser\TEST.BAT"
end_handler

# End of parameters for event definition "win_cmd_sample".
```

Definition File Options

The Definition File options used in this example are:

Option	Description
event_id	Identifier that uniquely identifies an event definition record.

event_type	Type of system event represented by the event definition record.
comp_name	Event-driven UEM Server responsible for monitoring the event.
state	Event definitions that should be processed or ignored by UEM.
inact_date_time	Date and time at which UEM will stop monitoring an event definition.
triggered_id	ID of an event handler record that UEM will execute when an event occurrence is triggered.
filespec	Name of a file to monitor.
rename_file	Specification for whether or not UEM should rename a monitored file when an event occurrence is triggered.
rename_filespec	Specification for how a file should be renamed when an event occurrence is triggered.
handler_id	Identifier that uniquely identifies an event handler record.
handler_type	Type of process executed for the event handler, based on the contents of the USER_COMMAND and USER_SCRIPT parameters.
maxrc	Highest value with which a handler can exit to still be considered as having executed successfully.
userid	ID of a user account in whose security context the handler process will be executed.
script_type	Type of script statements contained in the action field of the event handler record.
cmd	Command to execute on behalf of the event handler.

Components

UEMLoad Utility for Windows

Using a Stored Event Handler Record - UNIX

Using a Stored Event Handler Record in UNIX

In this example, a UEM Server (installed on a Windows system) will watch for the creation of a file called **uemtest.dat** in the **`C:\UEM Files`** directory.

If the file completes before the inactive time of *08:00* elapses, the event occurrence will be set to the **triggered** state. UEM then will execute the command or script contained in the event handler **h001**, which is the ID of a record in the event handler database.

If the file does not complete before the inactive time elapses, the event occurrence will be set to a **rejected** state. Since no event handler information is provided for a rejected occurrence, no further action will be taken by the UEM Server.

If the UEM Server does not detect the presence of **`C:\UEM Files\uemtest.dat`** before the inactive time of *08:00* elapses, the event will be set to an **expired** state.

> **Note**
> Because the inactive date value was omitted, UEM Manager will default the inactive date to the current date. Again, because no handler information is given for the **expired** state, no further action will be taken by the UEM Server once the event expires.

```
uem -host uemhost -event_type file
-filespec "C:\UEM Files\uemtest.dat"
-inact_date_time ,08:00 -userid uemuser -pwd uemusers_password
-triggered -handler_id h001
```

Command Line Options

The command line options used in this example are:

Option	Description
-host	List of one or more hosts upon which a command may run.
-event_type	Type of event to monitor.
-filespec	Name or pattern of the file whose creation should be detected and tracked for completion.
-inact_date_time	Date and time at which the state of the monitored event should be made inactive.
-userid	ID of a remote user account that the UEM Server uses to establish the security context in which event monitoring is performed.

-pwd	Password associated with -userid.
-triggered	Event state that, when encountered, will result in the execution of the associated event handler.
-handler_id	ID of a stored event handler record.

Components

Universal Event Monitor Manager for UNIX

Universal Event Monitor Server for Windows

Execute Script for Triggered Event Occurrence - UNIX

- Executing a Script for a Triggered Event Occurrence in UNIX
  - Command Line Options
  - Contents of Sample Script File
  - Components

Executing a Script for a Triggered Event Occurrence in UNIX

In this example, a UEM Server installed on a Windows machine will watch for the creation of a file called **uemtest.dat**. Since no path is specified, it will look for this file in the user's UEM Server working directory.

A relative inactive date/time is used to instruct the UEM Server to monitor the event for *10* minutes. If the file is detected and completes within that time, the event occurrence will be set to the **triggered** state. The script statements contained within the local file **`/UEMScripts/h_001.txt`** then will be written to a temporary script file on **uemhost** and executed by the UEM Server. The value specified by the **-handler_opts** option is appended to the command line constructed by UEM to execute the temporary script file. This will cause the values *parm1*, *parm2*, and *parm3* to be passed to the script. Any output generated by the script will be written to a file in the UEM Server working directory called **uemtest.log**.

If the file is detected, but does not complete before the inactive time elapses, the event occurrence will be set to a **rejected** state. Since no event handler information is provided for a rejected occurrence, no further action will be taken by the UEM Server.

If the UEM Server does not detect the presence of **uemtest.dat** before the inactive time elapses, the event will be set to an **expired** state. Again, because no handler information is given for this state, no further action will be taken by the UEM Server.

```
uem -host uemhost -event_type file -filespec uemtest.dat
-inact_date_time +10 -userid uemuser -pwd uemusers_password
-triggered -script /UEMScripts/h_001.txt
-handler_opts "parm1 parm2 parm3 >uemtest.log 2>&1"
```

Command Line Options

The command line options used in this example are:

Option	Description
-host	List of one or more hosts upon which a command may run.
-event_type	Type of event to monitor.
-filespec	Name or pattern of the file whose creation should be detected and tracked for completion.
-inact_date_time	Date and time at which the state of the monitored event should be made inactive.
-userid	ID of a remote user account that the UEM Server uses to establish the security context in which event monitoring is performed.
-pwd	Password associated with -userid.
-triggered	Event state that, when encountered, will result in the execution of the associated event handler.
-script	Complete path to a local script file that contains one or more system commands that should be executed on behalf of the event handler.

-handler_opts	Options that are passed as command line arguments to any process executed on behalf of an event handler.

Contents of Sample Script File

The following figure illustrates the contents of the **/UEMScripts/h_001.txt** file.

```
:: Sample script h_001.txt
@echo off

:: Program variables
set parmCtr=1

:: Loop through parameter list

:: **** Start of loop ****
:BeginLoop
if ""%1""=="""" goto EndLoop

:DisplayParm
echo Parm %parmCtr%: %1

:: Shift the next parm
shift
set /a parmCtr+=1

:: Go back to the top
goto BeginLoop

:: **** End of loop ****
:EndLoop
```

Components

Universal Event Monitor Manager for UNIX

Universal Event Monitor Server for Windows

Handling an Expired Event - UNIX

Handling an Expired Event in UNIX

In this example, a demand-driven UEM Server (installed on a different UNIX system) watches for the creation of a file called **uemtest.dat**. The **-filespec** option contains no path information, so UEM Server looks for this file in the home directory of **uemuser**.

A relative inactive date / time instructs the demand-driven Server to monitor the event for one (1) minute. If the UEM Server detects the file, and the file completes within that time, UEM sets the event occurrence to the triggered state. Since the command options contain no event handler information for a triggered occurrence, the UEM Server simply renames the file (by default). UEM Server then makes the event inactive, and ends.

If the UEM Server detects the file, but the file does not complete within 60 seconds, UEM sets the occurrence to the rejected state. Since the command options contain no event handler information for a rejected occurrence, the UEM Server leaves the file as-is and takes no further action.

If the UEM Server does not detect the presence of uemtest.dat before the inactive time elapses, it sets the event to the expired state. When this happens, the UEM Server executes the command specified by the **-cmd** option corresponding to the **-expired** option. In this example, UEM executes the **'ls -alR /uem files'** command.

Note
In this example, the -expired option is followed by the -options option, which redirects the output of the **'ls -alR "/uemfiles"'** command to a file in uemuser's home directory called **uemtest.log**.

```
uem -host uemhost -event_type file -filespec uemtest.dat
-userid uemuser -pwd uemusers_password
-inact_date_time +1
-expired -cmd 'ls -alR "/uem files"' -options '>uemtest.log 2>&1'
```

Command Line Options

The command line options used in this example are:

Option	Description
-host	List of one or more hosts upon which a command may run.
-event_type	Type of event to monitor.
-filespec	Name or pattern of the file whose creation should be detected and tracked for completion.
-userid	ID of a remote user account that the UEM Server uses to establish the security context in which event monitoring is performed.
-pwd	Password associated with -userid.
-inact_date_time	Date and time at which the state of the monitored event should be made inactive.
-expired	Event state that, when encountered, will result in the execution of the associated event handler.
-cmd	Complete path to an application file or remote script that should be executed on behalf of the event handler.
-options	Values that are passed as command line arguments to a particular handler specified for a given EVENT_STATE.

Components

Universal Event Monitor Manager for UNIX

Universal Event Monitor Server for UNIX

Add a Single Event Record - UNIX

Adding a Single Event Record for UNIX

In this example, a single event record identified as **payrollfile** is added to the local event definition database.

This event definition will instruct a UEM Server, which resides on the local (UNIX) system, to detect all occurrences of the file **`/tmp/payroll.dly`**. Whenever UEM detects this file and sets the associated event occurrence to a **triggered** state, UEM will execute the command or script contained in the stored event handler record that has an ID of **listdir**. If this event handler record does not exist at the time the event occurrence is triggered, an error will be issued by UEM.

When the record has been added to the event definition database, it is immediately available for use by a demand-driven UEM Server. In other words, there is no restriction with respect to how quickly a UEM Manager can reference the stored event definition after UEMLoad adds it to the database.

Because no values for the UEMLoad EVENT_STATE, ACTIVE_DATE_TIME, and INACTIVE_DATE_TIME options were specified, the default values of **enable**, the current date and time, and 2038.01.16,23:59, respectively, are used. This means the event will be monitored as soon as the event definition is assigned to an event-driven UEM Server. In this case, the event definition is assigned to the UEM Server component with an ID of **uems** (the default).

If this UEM Server component is active when the record is added, this assignment will occur the next time that the UEM Server refreshes its configuration. If the UEM Server component is not active, the assignment is made the next time it is started.

```
uemload -add -event_id payrollfile -event_type file
-filespec "/tmp/payroll.dly" -triggered_id listdir
```

Command Line Options

The command line options used in this example are:

Option	Description
-add	Writes one or more new event definition and/or event handler records to the appropriate database.
-event_id	Identifier that uniquely identifies an event definition record.
-event_type	Type of system event represented by the event definition record.
-filespec	Name of a file to monitor.
-triggered_id	ID of an event handler record that UEM will execute when an event occurrence is triggered.

Components

UEMLoad Utility for UNIX

Universal Event Monitor Server for UNIX

Add a Single Event Handler Record - UNIX

Adding a Single Event Handler Record for UNIX

In this example, a single handler record identified, **listdir**, is added to the local event handler database.

Whenever a UEM Server invokes this event handler, a handler process is started that executes the command **ls -al**, which lists the contents of the current directory on a UNIX system. The **encrypted.file** file, referenced by the **-encryptedfile** option, is a Universal Encrypted file. This file contains a user ID and, optionally, a password that is used by an event-driven UEM Server to establish a security context in which to execute the handler process (provided the USER_SECURITY option is enabled in the UEM Server configuration).

Once this record is added, it is available immediately to both demand-driven and event-driven UEM Servers.

Note
If a demand-driven UEM Server uses this handler, any user information specified in **encrypted.file** is overridden by the user information provided by the UEM Manager's command options.

```
uemload -add -handler_id listdir -encryptedfile encrypted.file
-cmd "ls -al"
```

Command Line Options

The command line options used in this example are:

Option	Description
-add	Writes one or more new event definition and/or event handler records to the appropriate database.
-handler_id	Identifier that uniquely identifies an event handler record.
-encryptedfile	Complete path to a file encrypted with Universal Encrypt.
-cmd	Command to execute on behalf of the event handler.

Components

UEMLoad Utility for UNIX

Universal Event Monitor Server for UNIX

Universal Encrypt

List All Event Definitions - UNIX

Listing All Event Definitions for UNIX

In this example, the **-list** option is used to dump all records in the event definition database and display them to **stdout**.

The asterisk (*****) must be escaped or enclosed in double quotation marks (that is: ***** or **"*"**, respectively).

```
uemload -list -event_id \*
```

Note
The default behavior when listing or exporting records, when neither an event ID nor a handler ID is specified, is to return all records. However, in this example above, even though no handler ID was specified, no event handler records are returned.

Conversely, if just a handler ID had been specified, no event definition records would be returned. Supplying an event ID and/or handler ID serves as a filter which causes **uemload** to return just those records specifically requested.

Command Line Options

The command line options used in this example are:

Option	Description
-list	Displays the complete contents of the specified event definition and/or event handler records.
-event_id	Identifier that uniquely identifies an event definition record.

Components

UEMLoad Utility for UNIX

List a Single Event Handler Record - UNIX

List a Single Event Handler Record for UNIX

In this example, the **-list** option is used to display the contents of an event handler record with an ID of **dirlist**.

```
uemload -list -handler_id dirlist
```

Command Line Options

The command line options used in this example are:

Option	Description
-list	Displays the complete contents of the specified event definition and/or event handler records.
-handler_id	Identifier that uniquely identifies an event handler record.

Sample List Output

The following figure illustrates sample output for this command. (The values shown are those that could be expected if the record were added using the command shown in Add a Single Event Handler Record - UNIX.)

In this specific instance, the user ID contained in **encrypted.file** (from Add a Single Event Handler Record - UNIX) is **sparkie**, and the record was added by the user account with an ID of **sbuser**.

```
UNV3659I Connecting to local broker on port 7887.
UNV3406I Universal Event Monitor Server component 1117035117 started.
UNV3666I Load request started at 11:32:04 AM 05/25/2011.
```

```
Event Handler(s):
=================
Handler ID.................: dirlist
Max Acceptable Return Code.: 0
User ID....................: sparkie
Command....................: ls -al
Last Modified On...........: 05/25/2011 11:32:06 AM
Last Modified By...........: sbuser

UNV3667I Universal Event Monitor Load is ending successfully with exit code
0.
```

Components

UEMLoad Utility for UNIX

Export Event Definition and Handler Databases - UNIX

Exporting the Event Definition and Event Handler Databases for UNIX

In this example, the -export option is used to dump all records in the event definition and event handler databases to a text file in the current directory named **uemout.txt**. This file is a UEMLoad definition file that also can be used to add or update records in the event definition and/or event handler databases.

The contents of the file resembles the examples shown in Definition File Format - UNIX.

```
uemload -export -deffile uemout.txt
```

Note
No event ID or handler ID is specified from the command line. If neither parameter is specified when listing or exporting records, the default behavior is to retrieve all database records.

Command Line Options

The command line options used in this example are:

Option	Description
-export	Dumps the contents of the specified event definition and/or event handler records to a text file that can be used as input to a subsequent run of the UEMLoad utility.
-deffile	Name of a file that contains event definition and/or event handler parameters.

Components

UEMLoad Utility for UNIX

List Event Definitions and Handlers Using Wildcards - UNIX

Listing Multiple Event Definitions and Event Handlers Using Wildcards for UNIX

In this example, the wildcards supported by **uemload** are demonstrated.

Wildcards can be used to select event definitions and event handlers whose respective IDs match the specified pattern.

- Asterisk (*****) can be used to match 0 or more characters.
- Question mark (**?**) can be used to match any single character.

All event definitions whose IDs start with the characters **event** are returned by the command below. In addition, all event handlers whose IDs begin with **handler0** and end with any two characters are selected.

```
uemload -list -event_id event* -handler_id handler0??
```

Command Line Options

The command line options used in this example are:

Option	Description
-list	Displays the complete contents of the specified event definition and/or event handler records.
-event_id	Identifier that uniquely identifies an event definition record.
-handler_id	Identifier that uniquely identifies an event handler record.

Components

UEMLoad Utility for UNIX

Add Record(s) Using Definition File - UNIX

Add Record(s) Using a Definition File for UNIX

In this example, a text file named **uemadd.txt** is used to add one or more records to the UEM databases. The contents of the file resemble those shown in Definition File Format - UNIX.

A definition file allows multiple records to be added to the event definition and/or event handler databases at the same time. When no definition file is used, only a single record can be added to the database(s).

```
uemload -add -deffile uemadd.txt
```

Command Line Options

The command line options used in this example are:

Option	Description
-add	Writes one or more new event definition and/or event handler records to the appropriate database.
-deffile	Name of a file that contains event definition and/or event handler parameters.

Components

UEMLoad Utility for UNIX

Add Record(s) Remotely Redirected from STDIN - UNIX

Add Record(s) Remotely, Using a Definition File Redirected from STDIN for UNIX

In this example, a definition load file named **uemadd.txt** is used to add one or more records to the databases of a remote UEM Server. The contents of the file resemble those shown in Definition File Format - UNIX.

Universal Command is used to execute UEMLoad on the remote UEM Server's system. The definition load file is redirected from standard input (stdin), which eliminates the step of copying the load file to the remote system before executing UEMLoad.

```
ucmd -cmd "uemload -add" -host rmthost -encryptedfile rmtacctinfo.enc
<uemadd.txt
```

Command Line Options

The command line options used in this example are:

Option	Description
-cmd	Remote command to execute.
-add	Writes one or more new event definition and/or event handler records to the appropriate database.
-host	List of one or more hosts upon which a command may run.
-encryptedfile	Encrypted command file.

Components

Universal Command Manager for UNIX

UEMLoad Utility for UNIX

Universal Event Monitor Server for UNIX

Add Record(s) Remotely Redirected from STDIN (for zOS) - UNIX

Add Record(s) Remotely, Using a Definition File Redirected from STDIN (for z/OS) for UNIX

In this example, a definition load file named **MY.UEM.DATA(UEMDEF)** is used to add one or more records to the databases of a remote UEM Server. The contents of the file resemble those shown in Definition File Format - UNIX.

Universal Command is used to execute UEMLoad on the remote UEM Server's system. It redirects standard input (stdin) from a data set allocated to the UNVIN ddname. This eliminates the step of copying the data set to the remote system before executing UEMLoad.

```
//STEP1    EXEC UCMDPRC
//UNVIN    DD  DISP=SHR,DSN=MY.UEM.DATA(UEMDEF)
//SYSIN    DD  *
-host                dallas
-userid              joe
-pwd                 ahzidaeh
-cmd                 "/opt/universal/bin/uemload -add"
/*
```

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-host	List of one or more hosts upon which a command may run.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-cmd	Remote command to execute.
-add	Writes one or more new event definition and/or event handler records to the appropriate database.

Components

Universal Command Manager for z/OS

UEMLoad Utility for UNIX

Universal Event Monitor Server for UNIX

Definition File Format - UNIX

Definition File Format for UNIX

The format of the definition load file for events and event handlers follows the keyword / value-pair convention used for Universal Agent configuration files. However, because more than one definition can be specified in a load file, some additional conventions are used.

- The **begin_event** and **end_event** keywords are used to mark the beginning and end, respectively, of each event definition entry.
- The **begin_handler** and **end_handler** keywords are used to mark the beginning and end, respectively, of each event handler entry.
- The **begin_script** and **end_script** keywords are used to mark the beginning and end, respectively, of any user script contained in the definition load file.

Lines that belong to the script must begin with the **stmt** keyword. Long **stmt** values that have to be split across lines can be done so using the **+** and **-** line continuation characters (as described in Configuration File Syntax). These lines will be accepted verbatim, and no script syntax validation will be done. Lines will continue to be added to the script until an **end_script**, **end_handler**, **begin_handler**, or **begin_event** keyword is read, or the end of the file is reached.

If a parameter's value contains spaces, it must be enclosed in single (**'**) or double (**"**) quotation marks.

If quotes are to be saved as part of the parameter's value, enclose the value in single (**'**) quotation marks quotes, and use a set of double (**"**) quotation marks to enclose the quoted value (for example, **optname 'optval1 "optval2 optval2a" optval3'**).

The **script** keyword can be used in lieu of a **begin_script/end_script** block, in which case the contents of the specified file will be written to the event handler.

A sample definition file for UNIX is shown in the following figure.

```
# Indented lines are for illustration only.  Leading spaces are
# ignored by UEMLoad.  Defaults will be used for any omitted
# values.

# Start of parameters for an event definition with an ID of
# "unix_event_sample".

begin_event
   event_id unix_event_sample
   event_type FILE
   comp_name uems
   state enable
   inact_date_time 2011.12.31,23:59
   triggered_id unix_script_sample
   filespec 'uem*.dat'
   rename_file yes
   rename_filespec '$(compname).$(compid).$(date).$(seqno)'
end_event

# End of parameters for event definition "unix_event_sample".

# Start of parameters for an event handler with an ID of
# "unix_script_sample".

begin_handler
   handler_id unix_script_sample
   handler_type SCRIPT
   maxrc 0
   userid uemuser
   begin_script
      stmt "#!/bin/sh"
      stmt ""
      stmt 'ls -al "/home/uem user"'
   end_script
   script_type bat
end_handler

# End of parameters for event handler "unix_script_sample".
```

```
# Start of parameters for an event definition with an ID of
# "unix_cmd_sample".

begin_handler
   handler_id unix_cmd_sample
   maxrc 0
   userid uemuser
   cmd 'homeuem usersomeapp'
end_handler

# End of parameters for event definition "unix_cmd_sample".
```

Definition File Options

The Definition File options used in this example are:

Option	Description
event_id	Identifier that uniquely identifies an event definition record.
event_type	Type of system event represented by the event definition record.
comp_name	Event-driven UEM Server responsible for monitoring the event.
state	Event definitions that should be processed or ignored by UEM.
inact_date_time	Date and time at which UEM will stop monitoring an event definition.
triggered_id	ID of an event handler record that UEM will execute when an event occurrence is triggered.
filespec	Name of a file to monitor.
rename_file	Specification for whether or not UEM should rename a monitored file when an event occurrence is triggered.
rename_filespec	Specification for how a file should be renamed when an event occurrence is triggered.
handler_id	Identifier that uniquely identifies an event handler record.
handler_type	Type of process executed for the event handler, based on the contents of the USER_COMMAND and USER_SCRIPT parameters.
maxrc	Highest value with which a handler can exit to still be considered as having executed successfully.
userid	ID of a user account in whose security context the handler process will be executed.
script_type	Type of script statements contained in the action field of the event handler record.
cmd	Command to execute on behalf of the event handler.

Components

UEMLoad Utility for UNIX

Fault Tolerance Implementation

Fault Tolerance Implementation

For Universal Agent, fault tolerance is the capability of its Universal Agent components to recover or restart from an array of error conditions that can occur in any large IT organization.

Errors occur as a result of human, software, or hardware conditions. The more resilient a product is to errors, the greater value it offers.

Currently, fault tolerance is implemented in three Universal Agent components:

- Universal Command
- Universal Connector
- Universal Data Mover

Detailed Information

The following pages provide detailed information for Fault Tolerance Implementation:

- Network Fault Tolerance - Universal Command
- Network Fault Tolerance - Universal Connector
- Network Fault Tolerance - Universal Data Mover
- Manager Fault Tolerance - Universal Command
- Client Fault Tolerance - Universal Connector
- Implementing Fault Tolerance - Examples

Additional Information

The following pages provide detailed information about terminating a z/OS job in a fault tolerant system:

- z/OS CANCEL Command Support

Network Fault Tolerance - Universal Command

- Overview
- Network Fault Tolerant Protocol
- Universal Command Manager
- Universal Command Server

Overview

Universal Command uses the TCP/IP protocol for communications over a data network. The TCP/IP protocol is a mature, robust protocol capable of resending packets and rerouting packets when network errors occur. However, data networks do have problems significant enough to prevent the TCP/IP protocol from recovering. As a result, the TCP/IP protocol terminates the connection between the application programs.

As with any application using TCP/IP, Universal Command is subject to these network errors. Should they occur, a product can no longer communicate and must shut down or restart. These types of errors normally show themselves as premature closes, connection resets, time-outs, or broken pipe errors.

Network Fault Tolerant Protocol

Universal Command provides the ability to circumvent network errors with its Network Fault Tolerant protocol. By using this protocol, Universal Command traps the connection termination caused by the network error and reestablishes the network connections. When connections have been reestablished, processing resumes automatically from the location of the last successful message exchange. No program restarts are required and no data is lost.

The Network Fault Tolerant protocol acknowledges successfully received messages and checkpoints successfully sent messages. This reduces data throughput. Consequentially, the use of network fault tolerance should be weighed carefully in terms of increased execution time versus the probability of network errors and cost of such errors. For example, it may be easier to restart a program then to incur increased execution time.

When a network connection terminates, the Universal Command Manager enters a network reconnect phase. In this phase, the Manager attempts to connect to the Universal Command Server and reestablish its network connections. The condition that caused the network error can persist only for seconds, or it can persist for days.

The Manager attempts Server reconnection for a limited amount of time, as specified by the following configuration options:

- RECONNECT_RETRY_COUNT (number of retry attempts)
- RECONNECT_RETRY_INTERVAL (frequency of retry attempts)

If all attempts fail, the Manager ends with an error.

When a network connection terminates, the Server's action depends on whether or not it is executing with Manager Fault Tolerance.

Without Manager Fault Tolerance, the Server enters a disconnected state and waits for the Manager to reconnect. The user process continues running. However, if the user process attempts any I/O on the standard files, it will block. The Server waits for the Manager to reconnect for a period of time defined by the Manager's RECONNECT_RETRY_COUNT and RECONNECT_RETRY_INTERVAL configuration options. When that time has expired, the Server terminates the user process and exits.

With Manager Fault Tolerance, the Server continues executing in a disconnected state. The Server satisfies all user process standard I/O requests. The user process does not block. It continues to execute normally. When the user process ends, the Server waits for a Manager reconnect for a period of time defined by the JOB_RETENTION configuration option.

Universal Command Manager

You can configure Universal Command Manager to request the use of the Network Fault Tolerant protocol via its NETWORK_FAULT_TOLERANT configuration option.

If the Server does not support the protocol or is not configured to accept the protocol, the Manager continues without using the protocol.

Universal Command Server

You can configure Universal Command Server with or without the Network Fault Tolerant protocol via its NETWORK_FAULT_TOLERANT configuration option.

If the Server is configured with the protocol off, the Manager cannot override it. If the Server is configured with the protocol on, the Manager NETWORK_FAULT_TOLERANT configuration option specifies whether or not the protocol is actually used.

Network Fault Tolerance - Universal Connector

- Overview
- Points of Failure
- Network Fault Tolerance Configuration Parameters

Overview

Universal Connector commands are processed by calling appropriate BAPI functions in the SAP system. The BAPI function calls are issued over an RFC connection. Universal Connector provides fault tolerance at the RFC level. If an RFC call fails, the call is retried until it completes successfully or exceeds a user-definable retry limit.

If an RFC call fails, Universal Connector will close the current RFC connection and establish a new RFC connection in order to continue processing. The process of establishing and preparing an RFC connection is referred to here as the RFC logon process. The RFC logon process involves establishing an RFC connection, logging on to the XMI interface and setting the XMI audit level. If the RFC logon process fails, it will be retried until it completes successfully, or exceeds a user definable retry limit. When the new RFC connection is established successfully, Universal Connector will reissue the failed RFC call.

The entire process of establishing a new RFC session and reissuing the failed RFC call will be retried until either:

- RFC call completes successfully.
- User-definable RFC retry limit is exceeded.

Some BAPI functions should not be retried in an unknown state; they are points of failure within the Universal Connector fault tolerant solution.

Points of Failure, below, lists the points of failure and their relationship to Universal Connector commands.

Points of Failure

The points of failure within Universal Connector fault tolerant architecture are:

- Job Submission
- Job Modification
- Job Start

Some BAPI functions called in these processes cannot be restarted in an unknown state without possible negative consequences. If an RFC call fails issuing those BAPIs, Universal Connector will end unsuccessfully.

Network Fault Tolerance Configuration Parameters

The following set of Universal Connector configuration options can be used to fine-tune the fault tolerance support for a particular environment:

- LISTEN_INTERVAL (-rfc_listen_interval)
- LOGON_RETRY_COUNT (-rfc_logon_retry_count)
- LOGON_RETRY_INTERVAL (-rfc_logon_retry_interval)
- SECURE_CFT (-rfc_retry_count)
- RETRY_CALL_INTERVAL (-rfc_retry_interval)
- TIMEOUT_INTERVAL (-rfc_timeout)

See RFC (Remote Function Call) Options for details concerning the use of these parameters.

Network Fault Tolerance - Universal Data Mover

- Overview
- Network Fault Tolerance

- Open Retry
- Component Management
- Communication State Values

Overview

For Universal Data Mover, fault tolerance is the capability of its components to recover or restart from an array of error conditions that occur in any large IT organization.

Errors occur as a result of human, software, or hardware conditions. The more resilient a product is to errors, the greater value it offers.

Currently, network fault tolerance is implemented in one Universal Data Mover component:

- Universal Data Mover

Network Fault Tolerance

UDM uses the TCP/IP protocol for communications over a data network. The TCP/IP protocol is a mature, robust protocol capable of resending and rerouting packets when network errors occur. However, data networks do have problems significant enough to prevent the TCP/IP protocol from recovering. As a result, the TCP/IP protocol terminates the connection between the application programs. Like any application using TCP /IP, UDM is subject to these network errors. Should they occur, a product can no longer communicate and must shutdown or restart. These types of errors normally show themselves as premature closes, connection resets, time-outs, or broken pipe errors.

UDM provides the ability to circumvent these types of errors with its Network Fault Tolerant protocol. By using the network fault tolerant protocol, UDM traps the connection termination caused by the network error and it reestablishes the network connections. Once connections are reestablished, processing automatically resumes from the location of the last successful message exchange. No program restarts are required and no data are lost.

The Network Fault Tolerant protocol acknowledges and checkpoints successfully received and sent messages, respectively. The network fault tolerant protocol does reduce data throughput. Consequentially, the use of network fault tolerance should be carefully weighed in terms of increased execution time versus the probability of network errors and cost of such errors. For example, it may be easier to restart a program then to incur increased execution time.

When a network connection terminates, the UDM Manager will enter a network reconnect phase. In the reconnect phase, the Manager attempts to connect to the UDM Server and reestablish its network connections. The condition that caused the network error can persist for seconds or days. The Manager will attempt Server reconnection for a limited amount of time (configured with the RECONNECT_RETRY_COUNT and RECONNECT_RETRY_INTERVAL configuration options). These two options specify, respectively, how many reconnect attempts are made and how often they are made. After all attempts have failed, the Manager ends with an error.

When a network connection terminates, the Server enters a disconnected state and waits for the Manager to reconnect. The user process continues running; however, if the user process attempts any I/O on the standard files, it will block. The Server waits for the Manager to reconnect for a period of time defined by the Manager's RECONNECT_RETRY_COUNT and RECONNECT_RETRY_INTERVAL configuration options. Once that time has expired, the Server terminates the user process and exits.

UDM can request the use of the Network Fault Tolerant protocol. If the Server does not support the protocol or is not configured to accept the protocol, the Manager continues without using the protocol.

The NETWORK_FAULT_TOLERANT and RECONNECT_RETRY_INTERVAL option is used to request the protocol.

Open Retry

Open Retry is a type of fault tolerance used at the session-establishment level.

(Network fault tolerance is used from the time that a session has been fully established until the session has terminated.)

Open Retry is used during the establishment phase of a session. UDM tries to establish a session when the open command is issued. If the OPEN_RETRY configuration option value is **yes**, and UDM fails to establish the session due to a network error, timeout, or the inability to start a transfer server, it will retry the open command based on the settings of the OPEN_RETRY_COUNT and OPEN_RETRY_INTERVAL configuration options.

Component Management

In order to fully understand UDM fault tolerant features, some understanding of how the Universal Broker manages components is necessary.

Universal Broker manages component start-up, execution, and termination. Universal Broker and its components have the ability to communicate service requests and status information between each other.

Universal Broker maintains a database of components that are active or have completed and waiting for restart or reconnection. The component information maintained by Universal Broker determines the current state of the component. This state information is required by Universal Broker to determine whether or not a restart or reconnect request from a Manager is acceptable. The ;Universal Broker component information can be viewed with the Universal Query utility.

One bit of component information maintained by Universal Broker is the component's communication state. The communication state primarily determines what state the Universal Data Mover Server is in regarding its network connection with a Manager and the completion of the user process and its associated spooled data.

Communication State Values

The following table describes the communication state values.

- **Reconnect** column indicates whether or not a network reconnect request is valid.
- **Restart** column indicates whether or not a restart request is valid.

State	Reconnect	Restart	Description
COMPLETED	NO	NO	Server and manager have completed. All standard output and standard error files have been sent to the manager and the user process's exit status.
DISCONNECTED	YES	YES	Server is not connected to the Manager. This occurs when a network error has occurred, the Manager halted, or the Manager host halted. The Server is either executing with the Network Fault Tolerant protocol, is restartable, or both. **Note** The Server cannot tell if the Manager is still executing or not, since it cannot communicate with the Manager.
ESTABLISHED	NO	NO	Server and Manager are connected and processing normally. This is the most common state when all is well.
RECONNECTING	NO	NO	Server has received a reconnect request from the Manager to recover a lost network connection. This state should not remain long, only for the time it takes to re-establish the network connections.
STARTED	NO	NO	Server has started. If the Server is restartable it is receiving the standard input file from the Manager and spooling it.

Manager Fault Tolerance - Universal Command

Overview

Distributed applications are comprised of many independent components running on host systems, throughout the enterprise, connected with a data network. Many of the host systems are in different physical locations, in different organization groups, and have different system management policies. It can be difficult to schedule individual host downtime when there are so many overlapping requirements.

Host systems must be shut down at scheduled intervals and, unfortunately, at unscheduled intervals. The impact of a system being down must be minimized by a distributed application.

With the Manager Fault Tolerant feature, Universal Command components (Managers and Servers) can be shut down and restarted at a later time. After a Manager has been started, it can be terminated and restarted at a later time. It can be shut down for any period of time: seconds, days, or even months.

Detailed Information

The following pages provide detailed information for Manager Fault Tolerance - Universal Command:

- Functionality
- Component Management

Manager Fault Tolerance - Universal Command - Functionality

- Manager Fault Tolerance Functionality
- Command Identifier
- Standard I/O Files
- Requesting Restart
- Case Example 1 - Normal Execution
  - Components
  - Sequence of Events
- Case Example 2 - Restart when User Process is Executing
  - Sequence of Events
- Case Example 3 - Restart when User Process has Ended
  - Sequence of Events

Manager Fault Tolerance Functionality

The basic functionality of Manager fault tolerance is:

1. Manager requests the execution of a command on a remote system.
2. Command executes on the remote system, optionally reading and writing data.
3. Manager redirects:
   - Its standard input data to the standard input of the remote command.
   - Standard output file and standard error file of the remote command to its own standard output and standard error.

If the Manager is terminated or the Manager's host system is shut down, the remote command cannot read the Manager's standard input or write its standard output and error files. Without Manager fault tolerance, the remote command must terminate, since its data source and destination are now gone. Otherwise, it would wait forever.

Manager fault tolerance provides an execution environment in which the Manager is not required in order for the user process to continue execution on the remote system. The user process can execute to completion with or without a Manager connected.

When the Manager starts a user process, the Manager executes as normal; standard output and standard error files are redirected back to the Manager as the user process produces the data. The difference is data spooling. In order for the user process to have real-time access to its input and output, the data is spooled in the Universal Spool Database. The spool provides complete independence from the Manager. The spool subsystem satisfies all data requirements for the user process via the Universal Command Server.

The Manager can terminate and a new Manager can restart and reconnect to the user process. If the user process has completed, the new Manager receives the user processes standard files and its exit status. The restarted Manager behaves in all ways as if it was the originating Manager .

Command Identifier

A Manager requests Manager fault tolerance with the MANAGER_FAULT_TOLERANT configuration option and by providing a command identifier (command ID) using the COMMAND_ID configuration option. The command ID identifies the unit of work being executed. In this context, a unit of work includes the Manager, Server, and user process.

The Manager indicates to the Server that this request is restartable. The COMMAND_ID configuration option provides a command identifier that uniquely identifies the Server and user process on the remote host. When a Manager is restarted, it must provide the same command ID identifying the Server and user process with which it wants to reconnect.

Providing a unique command ID is not trivial. Many Managers may be executing on many different hosts, and all executing work on the same Server host. It is possible for a Manager to start a restartable command from one host, terminate, and restart on a completely different host.

The command ID value can be any text value of unrestricted length. In practical terms, the character set and limits on command line length of the Manager host impose restrictions on the value.

Standard I/O Files

The Universal Spool system satisfies all user process data requests via the Universal Command Server. When the user process reads from its standard input file, the Server reads it from the spool and provides it to the user process. When the user process writes to standard output or error, the Server receives the data and writes it to the spool.

A Manager requesting restart capability (Manager fault tolerance) first transfers its entire standard input file to the Server, which it in turns writes to the spool. When all data has been received, the Server creates the user process. This provides complete Manager independence for the entire life of the user process.

As long as the Manager is connected, the standard output and standard error files are transferred to the Manager, as the user process produces the data, all in real-time. The data also is written to the spool. If the Manager terminates, the data is written to the spool only.

A restarted Manager is sent all of the standard output and standard error files, from the beginning, that currently is spooled. If the user process still is executing, the restarted Manager will receive all of the data currently spooled. When it has caught up with the data being produced, the Manager starts to receive the data from the user process as it is written.

Requesting Restart

When a restartable Manager is initiated, it is either an initial instance or a restarted instance of a command ID. The command ID identifies a unit of work represented by the Manager, Server, and user process. See #Command Identifier, above, for more information on the command ID.

The RESTART configuration option specifies whether or not the Manager instance is requesting a restart of a previous command ID. Possible RESTART values are **yes**, **no**, and **auto**.

The **auto** value specifies:

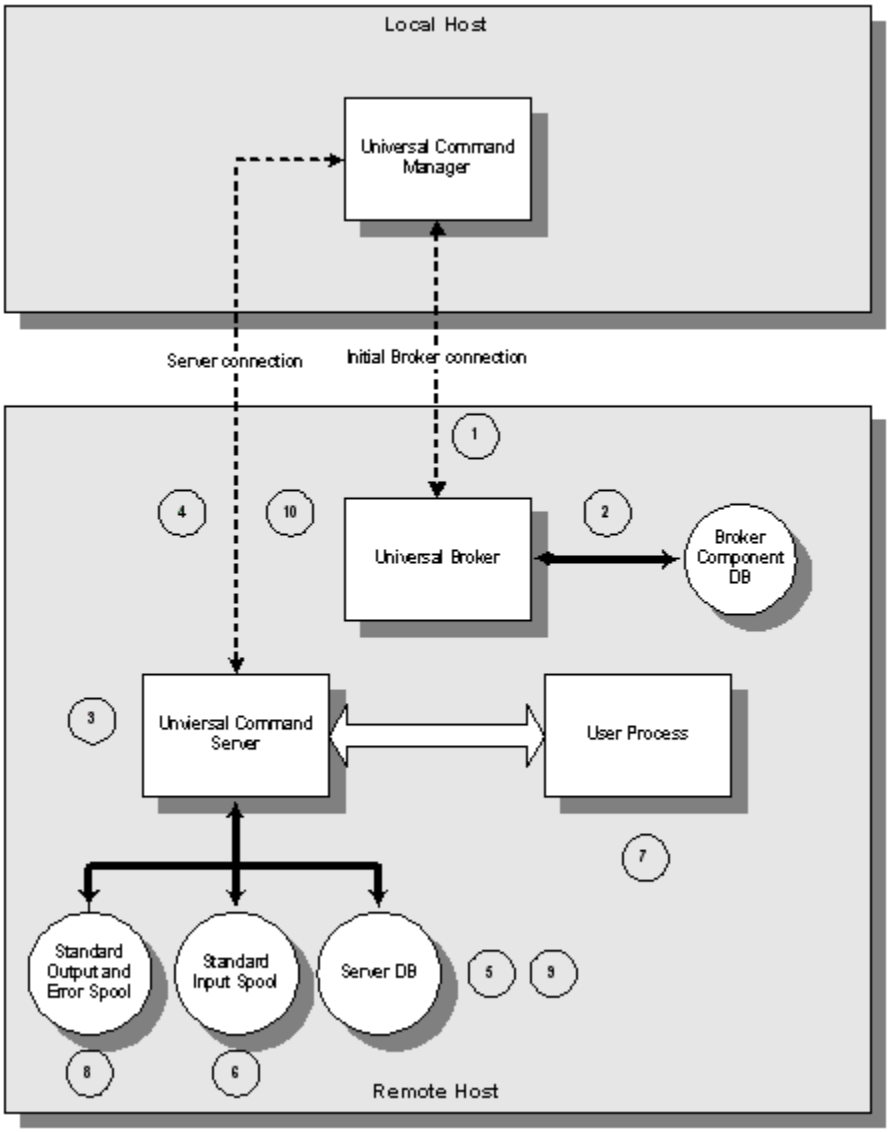
- If there is no existing command ID executing on the remote host, consider this Manager execution the first instance.
- If there is an existing command ID, and it is not connected to any Manager, consider this a restart of the command ID.

The **auto** value permits automatic restart by eliminating the need to modify the RESTART value for the initial instance and restarted instance.

Note
The **auto** value cannot be used with a COMMAND_ID configuration option value of *****, which specifies that the UCMD Manager will generate a unique command ID for each run.

Case Example 1 - Normal Execution

The following figure diagrams the sequence of events that occur when a restartable Manager requests the execution of a command on a remote host. In this case, the Manager and Server remain executing and connected until normal completion of the user process.



The Local Host is the host on which the Manager is being executed.

The Remote Host is the host on which the Manager is requesting command execution.

Components

The components involved are:

- Universal Command Manager
  The Manager requests remote execution of a command or script. The Manager executes the remote command in a manner such that the command appears to be executed locally.
- Universal Broker
  The Broker manages Universal Agent component execution.

- Universal Command Server
  The Server executes the Manager requested command and processes the user process's standard I/O requests.
- User Process
  The user process represents the Manager requested command.

Sequence of Events

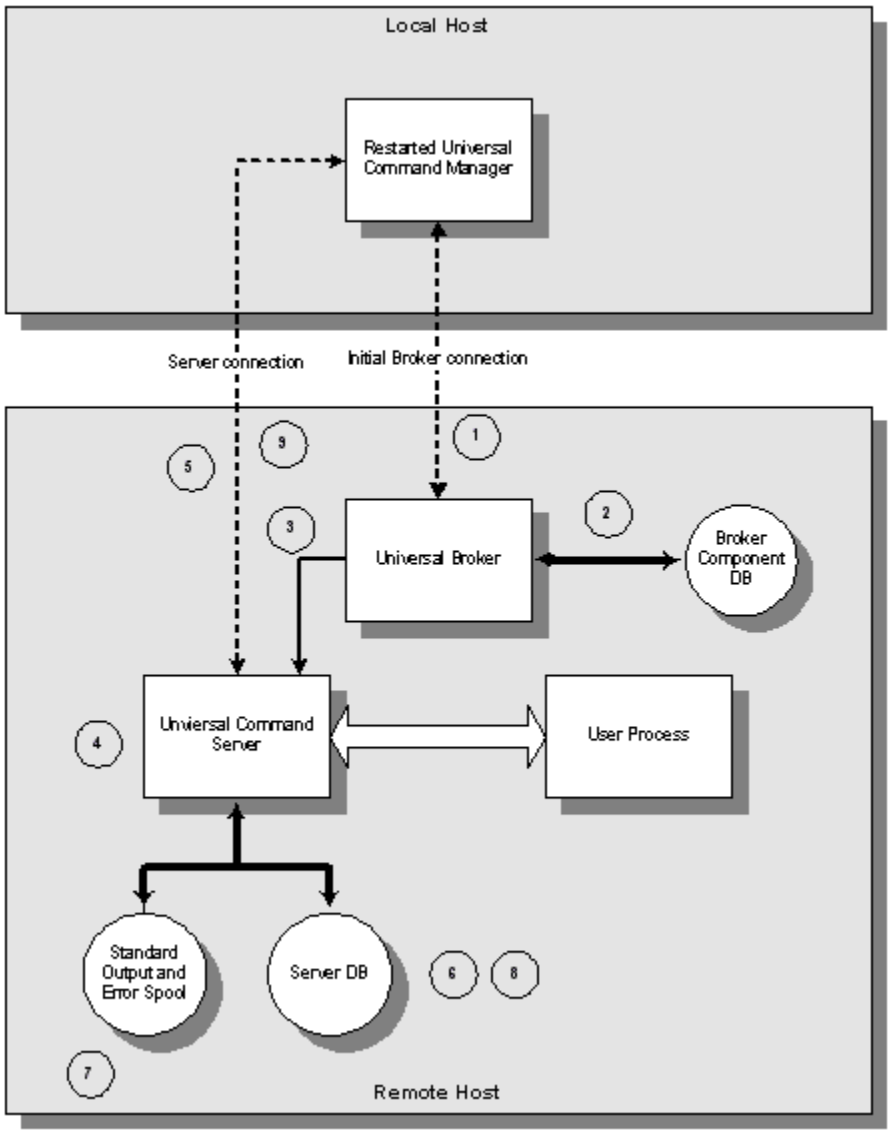
The diagram demonstrates the sequence of events that occur when a restartable Manager requests command execution on a remote host. The numbers enclosed in circles represent the sequence of events and correspond to the listed descriptions below.

Step 1	The Manager connects to the Broker and sends a request to start a Server. The start request from the Manager requests Manager fault tolerance and includes the command ID to identify the unit of work.
Step 2	The Broker records the unit of work in the Broker Component Database as restartable for possible future restarts.
Step 3	The Broker starts an instance of the Server.
Step 4	The Manager and Server exchange messages that specify all options used to carry out the request.
Step 5	The Server records the unit of work in the Universal Command Server Database for possible future restarts.
Step 6	The Manager sends all standard input data to the Server, and the Server writes the standard input data to the Universal Spool database.
Step 7	Once all standard input is spooled, the Server starts the user process.
Step 8	As the user process writes standard output and standard error data, the Server writes the data to the Universal Spool database. If the Manager is connected to the Server, the data is written to the Manager as well.
Step 9	The user process executes until completion. Once the user process completes, the Server writes the exit status of the user process to the Universal Command Server Database.
Step 10	The Server sends the exit status to the Manager. This completes the unit of work.

Case Example 2 - Restart when User Process is Executing

The following figure diagrams the sequence of events that occur when a Manager requests a restart of a currently executing unit of work. In this case the initial instance of the Manager terminated. A restarted instance of the Manager is started and requests to be reconnected to the unit of work.

This example continues from #Case Example 1 - Normal Execution; please refer to that example for details of the component descriptions included in the following diagram.



Sequence of Events

The diagram demonstrates the sequence of events that occur when a Manager requests to be restarted with a unit of work identified by a command ID. The numbers enclosed in circles represent the sequence of events and correspond to the listed descriptions below.

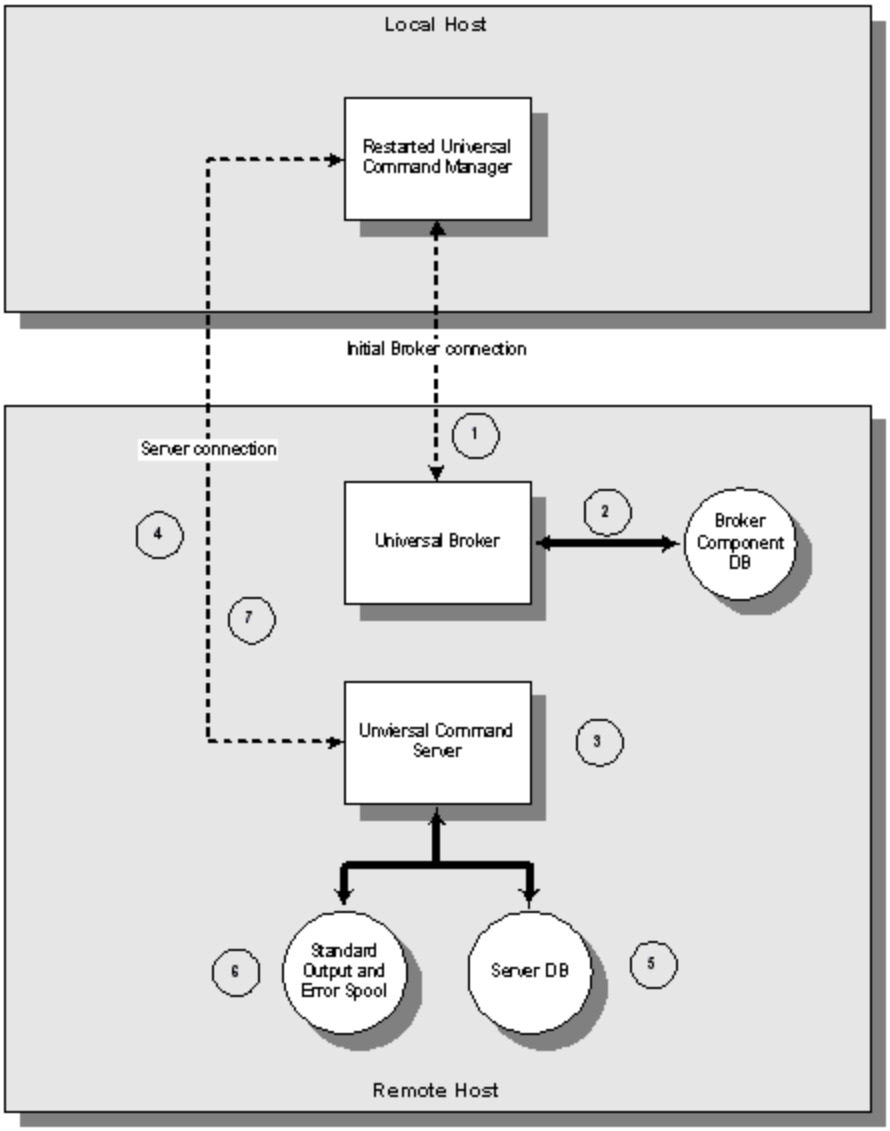
Step 1	The restarted instance of the Manager sends a restart request to the Broker. The restart request contains the command ID specified as part of the invocation of the Manager.
Step 2	The Broker verifies that the component is restartable and that the components communication state is acceptable for a restart request. If the Server component were currently connected to a Manager, its communication state would not permit a restart request.
Step 3	The Broker sends the restart request to the Server corresponding to the command ID.

Step 4	The Server authenticates the request with the Manager-supplied user ID and password. The password must be the same as the initial Manager instance.
Step 5	The Manager and Server exchange options that are used to carry out the request.
Step 6	The Server records the restart in the Universal Command Server Database.
Step 7	The Server sends spooled standard output and error files to the Manager. This is performed while the user process may still be writing standard output and error to the spool. Once all spooled output is sent to the Manager, the Server will send standard output and error from the user process as it is being produced.
Step 8	The user process executes to completion. The Server records the user process exit status in the Universal Command Server Database.
Step 9	The Server sends the exit status to the Manager. This completes the unit of work.

Case Example 3 - Restart when User Process has Ended

The following figure diagrams the sequence of events that occur when a Manager requests a restart of a unit of work that has completed. In this case, the initial instance of the Manager has terminated, the user process completed normally, and a restarted instance of the Manager is started and requests to be reconnected to the completed unit of work.

This example continues from #Case Example 1 - Normal Execution; please refer to that example for details of the component descriptions included in the following diagram.



Sequence of Events

The diagram demonstrates the sequence of events that occur when a Manager requests to be restarted with a unit of work identified by a command ID. The user process in this case has completed execution. The numbers enclosed in circles represent the sequence of events and correspond to the following descriptions:

Step 1	The restarted instance of the Manager sends a restart request to the Broker. The restart request contains the command ID specified as part of the invocation of the Manager.
Step 2	The Broker verifies that the component is restartable and that the components communication state is acceptable for a restart request. If the Server component were currently connected to a Manager, its communication state would not permit a restart request.
Step 3	Since the user process has completed, the Broker starts a new Server to process the restart request. The Server authenticates the request with the Manager-supplied user ID and password. The password must be the same as the initial Manager instance.
Step 4	The Manager and Server exchange options that are used to carry out the request.
Ste	The Server records the restart in the Universal Command Server Database.

Step 5	
Step 6	The Server sends spooled standard output and error files to the Manager.
Step 7	The Server sends the user process exit status to the Manager. This completes the unit of work.

Manager Fault Tolerance - Universal Command - Component Management

Overview

In order to fully understand Universal Command fault tolerant features, some understanding of how the Universal Broker manages components is necessary.

Universal Broker manages component start-up, execution, and termination. The Broker and its components have the ability to communicate service requests and status information between each other.

The Broker maintains a database of components that are active or have completed and waiting for restart or reconnection. The component information maintained by the Broker determines the current state of the component. This state information is required by the Broker to determine if a restart or reconnect request from a Manager is acceptable or not. The Broker's component information can be viewed with the Universal Query utility.

One piece of component information maintained by the Broker is the component's communication state. The communication state primarily determines what state the Server is in regarding its network connection with a Manager and the completion of the user process and its associated spooled data.

Communication State Values

The following table describes the communication state values.

- **Reconnect** column indicates whether or not a network reconnect request is valid.
- **Restart** column indicates whether or not a restart request is valid.

State	Reconnect	Restart	Description
STARTED	NO	NO	Server has started. If the Server is restartable, it is receiving the standard input file from the Manager and spooling it.
ESTABLISHED	NO	NO	Server and Manager are connected and processing normally. This is the most common state when all is well.
DISCONNECTED	YES	YES	Server is not connected to the Manager. This occurs when a network error has occurred, the Manager halted, or the Manager host halted. The Server is either executing with the Network Fault Tolerant protocol, is restartable, or both. **Note** The Server cannot tell whether or not the Manager is still executing, since it cannot communicate with the Manager.
ORPHANED	NO	YES	Manager has terminated after sending a termination message to the Server to notify it of its termination. This state only occurs if the Server is restartable.
RECONNECTING	NO	NO	Server has received a reconnect request from the Manager to recover a lost network connection. This state should not remain long; only for the time it takes to re-establish the network connections.
RESTARTING	NO	NO	Server has received a restart request from the Manager. This state should not remain long; only for the time it takes to re-establish network connections.
PENDING	NO	YES	A restartable Server and its user process have completed. The user process standard output and error files are in the spool. A Manager has not been restarted to pick up the spooled files and user process exit status. The Server remains in this state until a Manager is restarted.
COMPLETED	NO	NO	Server and Manager have completed. All standard output and standard error files have been sent to the Manager and the user process's exit status.

Client Fault Tolerance - Universal Connector

Introduction

The Client Fault Tolerance feature allows the Universal Connector client application to be shut down and restarted at a later time.

The following pages provide detailed information for Client Fault Tolerance - Universal Connector:

- Client Fault Tolerance - Universal Connector Jobs
  - Modes
  - Parameters
  - Command ID Job Step
  - Command Identifier
  - Requesting Restart
- Client Fault Tolerance - Universal Connector Process Chains
  - Modes
  - Parameters
  - Command ID Job Step
  - Command Identifier
  - Requesting Restart

Client Fault Tolerance - Universal Connector Jobs

Overview

The Client Fault Tolerance feature allows the Universal Connector client application to be shut down and restarted at a later time.

This functionality helps avoid problems that can result if the Universal Connector application terminates unexpectedly while processing an SAP job. In such an instance, the Client Fault Tolerance restart capability allows Universal Connector to reconnect to a running (or completed) job while preventing the unintentional start of a new instance of the original SAP job.

To achieve Client Fault Tolerance, Universal Connector must be able to associate the SAP jobs it defines and starts with a particular unit of work. In this context, a unit of work includes the Universal Connector client and SAP job instance.

To associate an SAP job with a particular unit of work, the user must be able to specify some identifying characteristic that is specific to that unit of work. The SAP system uniquely identifies job instances by a job name/job ID pair. Since the job name must be reusable and the job ID is assigned by the SAP system at the time of definition, Universal Connector must use an alternative job characteristic. This alternative job characteristic is the Universal Connector command Identifier.

Universal Connector references a particular unit of work by a job name/Command Identifier combination. The Universal Connector command identifier is tied to the SAP job by appending a Command ID Job Step to the SAP job associated with the Universal Connector command instance. The Command ID job step is required for identification purposes only. Therefore, the program used for the Command ID step is intended to add minimum overhead to the job. The Command ID used for the job is included in the definition of the Command ID job step.

Detailed Information

The following pages provide detailed information for Client Fault Tolerance - Universal Connector Jobs:

- Modes
- Parameters
- Command ID Job Step
- Command Identifier
- Requesting Restart

Client Fault Tolerance - Universal Connector Jobs - Modes

- Overview
- Secure Client Fault Tolerance (Secure CFT) Mode
- Pre-XBP 2.0 Client Fault Tolerance (CFT) Mode

Overview

Universal Connector supports two modes of client fault tolerance:

1. Secure Client Fault Tolerance (Secure CFT)
2. Pre-XBP 2.0 Client Fault Tolerance (CFT)

The mode of client fault tolerance to be used by Universal Connector is determined by the value of the SECURE_CFT option.

Valid values for this option are **yes** and **no**:

- **yes** will cause Universal Connector to use the Secure CFT mode.
- **no** will cause Universal Connector to use the original Pre-XBP 2.0 CFT mode.

The default value is **yes**.

Both modes of CFT follow the same basic process flow. When Universal Connector is requested to restart a particular command ID job, it queries the SAP system for all jobs with the specified job name. The list of jobs returned by the SAP system is scanned for a job that contains an appropriate Command ID Job Step. If found, Universal Connector will re-connect to the SAP job instance and satisfy the command line requirements.

Universal Connector is capable of restarting a command ID as long as the associated command ID job remains in the SAP system.

Secure Client Fault Tolerance (Secure CFT) Mode

This mode is an enhancement of the original implementation. The secure CFT mode requires XBP 2.0 to be installed on the SAP side of the Universal Connector connection. In this mode, an ABAP program step is used for the command ID job step.

Using an ABAP program step as the Command ID job step eliminates the security and ease of use drawbacks mentioned above for external program job steps.

- Security
  The execution of ABAP programs and the resources required by them are secured by SAP authorization checks.
- Ease of Use
  ABAP program job steps do not require a target host. They run on whichever application server the job runs on. Therefore, there are no target specific parameters required for secure CFT mode.

Pre-XBP 2.0 Client Fault Tolerance (CFT) Mode

This mode is the original implementation of client fault tolerance used prior to the release of XBP 2.0. Due to limitations in the XBP 1.0 interface, Universal Connector client fault tolerance on pre-XBP 2.0 SAP systems uses an external program step as the command ID job step.

Using an external program step as the command ID job step has the following security and ease of use drawbacks:

- Security drawback
  Using an external program job step requires the SAP user ID to have authority to run external programs. This authority cannot be given lightly for the following reason: When running an external job step, the SAP system first performs an authorization check to see if the user ID has the right to run an external program. If so, the external program is run under the user ID of the user who started the SAP system
- Ease of use drawback
  Using an external program job step requires a target host be specified for the external program to run on. This requires information about the SAP landscape that may not be readily available. Also, this presents the possibility of the Universal Connector job's parameters becoming out of sync with the SAP landscape.

Client Fault Tolerance - Universal Connector Jobs - Parameters

- Client Fault Tolerance Target Host
- Client Fault Tolerance Command Prefix
- Secure Client Fault Tolerance Option
- Client Fault Tolerance ABAP Program

Client Fault Tolerance Target Host

The client fault tolerance target host parameter is only required for pre-XBP 2.0 client fault tolerance mode. If the secure CFT mode is being used, the client fault tolerance target host parameter is ignored.

As part of an external program command ID job step definition, SAP requires a target host on which to run the external program (echo). Universal Connector provides the client fault tolerance target host parameter to specify the target host for the command ID job step.

The Client Fault Tolerance Target Host is specified with the CFT_TARGET_HOST option.

Client Fault Tolerance Command Prefix

The client fault tolerance command prefix parameter is only required for pre-XBP 2.0 client fault tolerance mode. If the secure CFT mode is being used, the client fault tolerance command prefix parameter is ignored.

The external program command ID job step has the potential to be run on any operating system reachable by the SAP system. The operating system that the Command ID Job Step runs on is that which exists on the host system specified by the Client Fault Tolerance Target Host parameter. Different operating systems may require commands to be called in different ways. Therefore, the Client Fault Tolerance Command Prefix parameter allows the user to specify the prefix necessary to run the echo command on the host system specified by the Client Fault Tolerance Target Host parameter.

For example, to run the echo command on a Windows system, the following command line entry would be required for an SAP external job step: **cmd /C echo**. Therefore, the Client Fault Tolerance Command Prefix for this system would be: **cmd /C**.

The Client Fault Tolerance Command Prefix parameter is specified with the CFT_COMMAND_PREFIX option.

Secure Client Fault Tolerance Option

The mode of client fault tolerance to be used by Universal Connector is determined by the value of the SECURE_CFT option.

Valid values for this option are **yes** and **no**:

- **yes** will cause Universal Connector to use the Secure CFT mode.
- **no** will cause Universal Connector to use the original Pre-XBP 2.0 CFT mode.

The default value is **yes**.

Client Fault Tolerance ABAP Program

The client fault tolerance ABAP program parameter is only required for secure CFT mode. If the pre-XBP 2.0 CFT mode is being used, the client fault tolerance ABAP program parameter is ignored.

The client fault tolerance ABAP program parameter is used to specify the ABAP program to use for the command ID job step. Any ABAP program can be specified. The Universal Connector internal default ABAP program to use is BTCTEST. BTCTEST is a standard SAP ABAP program that should be available on all SAP systems. It does no real processing and can be considered a dummy program that does not interfere with job processing and places little overhead on the system.

Client Fault Tolerance - Universal Connector Jobs - Command ID Job Step

- Overview
  - Pre-XBP 2.0 CFT Mode
  - Secure CFT Mode

Overview

Universal Connector creates Command ID jobs by appending a job step to the user's SAP job being defined to the system. This appended job step is the Universal Connector Command ID Job Step.

Pre-XBP 2.0 CFT Mode

In pre-XBP 2.0 CFT mode, the Universal Connector Command ID Job Step executes the external program echo. A string containing the command ID is inserted in the parameter field of the job step. The echo command is lightweight, does not interfere with the original job, and results in the command ID being printed to the joblog.

Secure CFT Mode

In secure CFT mode, the Universal Connector command ID job step executes an ABAP program. The ABAP program defined to the command id step is user configurable with the covetable parameter. Any ABAP program can be specified. The Universal Connector internal default ABAP program to use is BTCTEST. BTCTEST is a standard SAP ABAP program that should be available on all SAP systems. It does no real processing and can be considered a dummy program that does not interfere with job processing and places little overhead on the system.

Client Fault Tolerance - Universal Connector Jobs - Command Identifier

Command Identifier

Universal Connector requests client fault tolerance by providing a command identifier. The command identifier is specified on the command line with parameter **-cmdid**. The command ID/job name pair identifies the unit of work being executed.

The command ID option provides a command identifier that (paired with job name) uniquely identifies the SAP job on the SAP system. When a Universal Connector job is restarted, it must provide the same command ID identifying the SAP job with which it wants to reconnect.

Providing a unique command ID is not trivial. Many Universal Connector clients may be executing on many different hosts, all executing work on the same SAP system. It is possible for a Universal Connector client to start a restartable job from one host, terminate, and restart on a completely different host.

The command ID value can be any text value up to 245 characters in length. In practical terms, the character set and limits on command line length of the Universal Connector host may impose further restrictions on the value.

Client Fault Tolerance - Universal Connector Jobs - Requesting Restart

- Requesting Restart
- Controlling Auto Restart

Requesting Restart

When a restartable Universal Connector command is initiated, it is either an initial instance or a restarted instance of a command ID.

The RESTART option is specified on the command line with parameter **-restart**. RESTART specifies whether or not the Universal Connector command instance is requesting a restart of a previous command ID. Possible RESTART values are **yes**, **no**, or **auto**.

The **auto** value specifies that if there is no existing command ID job on the SAP system, consider this Universal Connector execution the first instance. If there is an existing command ID job, consider this a restart of the command ID. The **auto** value permits automatic restart by eliminating the need to modify the RESTART value for the initial instance and restarted instance.

It is important to note that when using the RESTART **auto** value, Universal Connector will not start a new instance of a job on the SAP system if a job matching the job name/command ID exists in the SAP system. Universal Connector will continue to reconnect to the existing SAP job.

Without considering the behavior resulting from the use of RESTART **auto**, it may be possible for one to assume that a job has been run multiple times when, in fact, Universal Connector has been reconnecting to the same job instance. Informational messages are printed by Universal Connector to standard error to indicate the reconnected status but, if the message level is not set to **info**, the messages will not be seen.

Controlling Auto Restart

Misunderstanding the auto restart behavior described above can potentially have serious consequences. For this reason, the ALLOW_AUTO_RESTART lets you allow or disallow the use of auto restart. You can make this specification in the Universal Connector configuration file and override it on the command line.

Client Fault Tolerance - Universal Connector Process Chains

Overview

The Client Fault Tolerance feature allows the Universal Connector client application to be shut down and restarted at a later time.

This functionality helps avoid problems that can result if the Universal Connector application terminates unexpectedly while processing an SAP process chain. In such an instance, the Client Fault Tolerance restart capability allows Universal Connector to reconnect to a running (or completed) process chain while preventing the unintentional start of a new process chain instance.

To achieve Client Fault Tolerance, Universal Connector must be able to associate an SAP process chain instance with a particular unit of work. In this context, a unit of work includes the Universal Connector client and SAP process chain instance.

To associate an SAP process chain instance with a particular unit of work, the user must be able to specify some identifying characteristic that is specific to that unit of work. The SAP system uniquely identifies process chains by a log ID. However, since the process chain log ID is assigned by the SAP system at the time of instantiation, Universal Connector must use an alternative characteristic; the Universal Connector command Identifier.

Universal Connector references a particular unit of work by a chain ID/Command Identifier combination. The Universal Connector command identifier is associated with the SAP process chain by creating a dummy job in the SAP system with a log ID step and a Command ID Step. The log ID step stores the value of the process chain log ID that is generated by the SAP system. The Command ID step stores the value of the Universal Command ID. The dummy job that is created for client fault tolerant process chains is never actually run. It is just used to associate a Universal Connector Command ID with an SAP process chain log ID within the SAP system. After successful processing of the process chain, the dummy job is removed from the system.

Detailed Information

The following pages provide detailed information for Client Fault Tolerance - Universal Connector Process Chains:

- Modes
- Parameters
- Command ID Job Step
- Command Identifier
- Requesting Restart

Client Fault Tolerance - Universal Connector Process Chains - Modes

- Overview
- Secure Client Fault Tolerance (Secure CFT) Mode
- Pre-XBP 2.0 Client Fault Tolerance (CFT) Mode

Overview

Universal Connector supports two modes of client fault tolerance:

1. Secure Client Fault Tolerance (Secure CFT)
2. Pre-XBP 2.0 Client Fault Tolerance (CFT)

The mode of client fault tolerance to be used by Universal Connector is determined by the value of the SECURE_CFT option.

Valid values for this option are **yes** and **no**:

- **yes** will cause Universal Connector to use the Secure CFT mode.
- **no** will cause Universal Connector to use the original Pre-XBP 2.0 CFT mode.

The default value is **yes**.

Both modes of CFT follow the same basic process flow. When Universal Connector is requested to restart a particular command ID process chain, it queries the SAP system for all jobs with the specified job name (the chain ID is used for the job name). The list of jobs returned by the SAP system is scanned for a job that contains an appropriate Command ID Job Step. If found, Universal Connector will re-connect to the SAP process chain instance associated with the job and satisfy the command line requirements.

Universal Connector is capable of restarting a command ID as long as the associated command ID job remains in the SAP system.

Secure Client Fault Tolerance (Secure CFT) Mode

This mode is an enhancement of the original implementation. The secure CFT mode requires XBP 2.0 to be installed on the SAP side of the Universal Connector connection. In this mode, an ABAP program job step is used for the log ID step and the command ID step.

Using an ABAP program job step as the log ID and Command ID steps eliminates the security and ease of use drawbacks mentioned above for external program steps.

- Security
  The execution of ABAP programs and the resources required by them are secured by SAP authorization checks.
- Ease of Use
  ABAP program job steps do not require a target host. They run on whichever application server the job runs on. Therefore, there are no target specific parameters required for secure CFT mode.

Pre-XBP 2.0 Client Fault Tolerance (CFT) Mode

This mode is the original implementation of client fault tolerance used prior to the release of XBP 2.0. Due to limitations in the XBP 1.0 interface, Universal Connector client fault tolerance on pre-XBP 2.0 SAP systems uses an external program step as the log ID and command ID steps.

Using external program steps as the log ID and command ID steps has the following security and ease of use drawbacks:

- Security drawback
  The dummy job used for client fault tolerance with process chains is never actually run by the Universal Connector so, there is no security drawback.
- Ease of use drawback
  Using an external program step requires a target host be specified for the external program to run on. This requires information about the SAP landscape that may not be readily available. Also, this presents the possibility of the Universal Connector job's parameters becoming out of sync with the SAP landscape.

Client Fault Tolerance - Universal Connector Process Chains - Parameters

- Client Fault Tolerance Target Host
- Client Fault Tolerance Command Prefix
- Secure Client Fault Tolerance Option
- Client Fault Tolerance ABAP Program

Client Fault Tolerance Target Host

The client fault tolerance target host parameter is only required for pre-XBP 2.0 client fault tolerance mode. If the secure CFT mode is being used, the client fault tolerance target host parameter is ignored.

As part of an external program command ID step definition, SAP requires a target host on which to run the external program (echo). Universal Connector provides the client fault tolerance target host parameter to specify the target host for the command ID step.

The Client Fault Tolerance Target Host is specified with the CFT_TARGET_HOST option.

Client Fault Tolerance Command Prefix

The client fault tolerance command prefix parameter is only required for pre-XBP 2.0 client fault tolerance mode. If the secure CFT mode is being used, the client fault tolerance command prefix parameter is ignored.

The external program command ID step has the potential to be run on any operating system reachable by the SAP system. The operating system that the Command ID Job Step runs on is that which exists on the host system specified by the Client Fault Tolerance Target Host parameter. Different operating systems may require commands to be called in different ways. Therefore, the Client Fault Tolerance Command Prefix parameter allows the user to specify the prefix necessary to run the echo command on the host system specified by the Client Fault Tolerance Target Host parameter.

For example, to run the echo command on a Windows system, the following command line entry would be required for an SAP external step: **cmd /C echo**. Therefore, the Client Fault Tolerance Command Prefix for this system would be: **cmd /C**.

The Client Fault Tolerance Command Prefix parameter is specified with the CFT_COMMAND_PREFIX option.

Secure Client Fault Tolerance Option

The mode of client fault tolerance to be used by Universal Connector is determined by the value of the SECURE_CFT option.

Valid values for this option are **yes** and **no**:

- **yes** will cause Universal Connector to use the Secure CFT mode.
- **no** will cause Universal Connector to use the original Pre-XBP 2.0 CFT mode.

The default value is **yes**.

Client Fault Tolerance ABAP Program

The client fault tolerance ABAP program parameter is only required for secure CFT mode. If the pre-XBP 2.0 CFT mode is being used, the client fault tolerance ABAP program parameter is ignored.

The client fault tolerance ABAP program parameter is used to specify the ABAP program to use for the command ID step. Any ABAP program can be specified. The Universal Connector internal default ABAP program to use is BTCTEST. BTCTEST is a standard SAP ABAP program that should be available on all SAP systems. It does no real processing and can be considered a dummy program that does not interfere with job processing and places little overhead on the system.

Client Fault Tolerance - Universal Connector Process Chains - Dummy Job with Log ID and Command ID Job Steps

- Overview
  - Pre-XBP 2.0 CFT Mode
  - Secure CFT Mode

Overview

Universal Connector creates a dummy Command ID job using the process chain's chain ID as a job name.

Two steps are created for the job:

1. Log ID step to store the log ID of the associated process chain instance.
2. Command ID step to store the Universal Connector Command ID.

The dummy job is never started by Universal Connector. It is just used to associate the Universal Command ID process chain Unit of work with a specific SAP process chain instance.

Pre-XBP 2.0 CFT Mode

In pre-XBP 2.0 CFT mode, the Universal Connector Log ID and Command ID Job Steps are defined as external program steps. A string containing the process chain log ID is inserted in the parameter field of the log ID step and a string containing the command ID is inserted in the parameter field of the Command ID step.

Secure CFT Mode

In secure CFT mode, the Universal Connector log ID and command ID steps are defined as ABAP job steps. The ABAP program defined to the log ID and command ID steps is user configurable. Any ABAP program can be specified. The Universal Connector internal default ABAP program to use is BTCTEST. BTCTEST is a standard SAP ABAP program that should be available on all SAP systems. It does no real processing and can be considered a dummy program that does not interfere with job processing and places little overhead on the system.

Client Fault Tolerance - Universal Connector Process Chains - Command Identifier

Command Identifier

Universal Connector requests client fault tolerance by providing a command identifier. The command identifier is specified on the command line with parameter **-cmdid**. The command ID/chain ID pair identifies the unit of work being executed.

The command ID option provides a command identifier that (paired with chain ID) uniquely identifies the SAP process chain on the SAP system. When a Universal Connector job is restarted, it must provide the same command ID identifying the SAP process chain with which it wants to reconnect.

Providing a unique command ID is not trivial. Many Universal Connector clients may be executing on many different hosts, all executing work on the same SAP system. It is possible for a Universal Connector client to start a restartable job from one host, terminate, and restart on a completely different host.

The command ID value can be any text value up to 245 characters in length. In practical terms, the character set and limits on command line length of the Universal Connector host may impose further restrictions on the value.

Client Fault Tolerance - Universal Connector Process Chains - Requesting Restart

- Requesting Restart
- Controlling Auto Restart

Requesting Restart

When a restartable Universal Connector command is initiated, it is either an initial instance or a restarted instance of a command ID.

The RESTART option is specified on the command line with parameter **-restart**. RESTART specifies whether or not the Universal Connector command instance is requesting a restart of a previous command ID. Possible RESTART values are **yes**, **no**, or **auto**.

The **auto** value specifies that if there is no existing command ID job on the SAP system, consider this Universal Connector execution the first instance. If there is an existing command ID job, consider this a restart of the command ID. The **auto** value permits automatic restart by eliminating the need to modify the RESTART value for the initial instance and restarted instance.

It is important to note that when using the RESTART **auto** value, Universal Connector will not start a new instance of a process chain on the SAP system if a job matching the chain ID/command ID exists in the SAP system. Universal Connector will continue to reconnect to the existing SAP process chain.

Without considering the behavior resulting from the use of RESTART **auto**, it may be possible for one to assume that a process chain has been run multiple times when, in fact, Universal Connector has been reconnecting to the same process chain instance. Informational messages are printed by Universal Connector to standard error to indicate the reconnected status but, if the message level is not set to **info**, the messages will not be seen.

It is also important to note that in the context of client fault tolerant process chain commands, the term **RESTART** refers to the logical unit of work that involves the Universal Connector and the SAP process chain instance. A restart of this logical unit of work does not necessarily restart the process chain instance on the SAP system. The restart alone is actually a "reconnect" to the process chain instance. In order to actually restart the process chain instance on the SAP system, the FORCE configuration option must be added to the command line (-force yes).

Controlling Auto Restart

Misunderstanding the auto restart behavior described above can potentially have serious consequences. For this reason, the ALLOW_AUTO_RESTART lets you allow or disallow the use of auto restart. You can make this specification in the Universal Connector configuration file and override it on the command line.

Sample Command Lines For Working With Client Fault Tolerance

Sample Command Lines For Working With Client Fault Tolerance

- Working With Job Definition Files
- Working With Pre-defined SAP Jobs

Working With Job Definition Files

- Initial Run of a Command ID Job
  - Pre-XBP 2.0 CFT Mode
  - Secure CFT Mode
  - Command Line Options
- Restart of a Command ID Job
  - Pre-XBP 2.0 CFT Mode
  - Secure CFT Mode
  - Command Line Options
- Run a Command ID Job Using Restart AUTO
  - Pre-XBP 2.0 CFT Mode
  - Secure CFT Mode
  - Command Line Options

Initial Run of a Command ID Job

The following examples will submit, start, and wait for the command ID job defined in job definition file **jobdef**. Because the RESTART option is set to **no**, Universal Connector will scan the SAP system to ensure that a command ID job with the same job name/command ID pair does not already exist on the system.

If a matching command ID job is found on the SAP system, Universal Connector will exit with an error code before performing the job submission.

Pre-XBP 2.0 CFT Mode

```
usap -userid bob -pwd secret -submit jobdef -start -wait -cft_secure_cft no
     -cft_target_host pwdf0196 -cft_cmd_prefix "cmd /C" -cmdid E8E8E80001 -
restart no
```

Note
The Client Fault Tolerance Command ID Prefix (-cft_cmd_prefix) is set up for a Windows host. In many user environments, the Client Fault Tolerance Command ID Prefix option can be specified in the configuration file and will never need to be specified on the command line. The same is true for the Client Fault Tolerance Target Host option (-cft_target_host). The secure CFT option (-cft_secure_cft) also would be set in the configuration file in most cases.

Secure CFT Mode

```
usap -userid bob -pwd secret -submit jobdef -start -wait -cft_secure_cft yes
     -cft_abap BTCTEST -cmdid E8E8E80001 -restart no
```

Note
In secure CFT mode, the (-cft_secure_cft) and (-cft_abap) parameters would most likely be specified in the Universal Connector configuration file.

Command Line Options

Command line options used in these examples are:

Command Options	Description
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-submit	Defines a job to the SAP system.
-start	Starts the newly defined job.
-wait	Wait for the SAP job to complete processing.
-cft_secure_cft	Mode of client fault tolerance to be used for the command invocation.
-cft_target_host	In pre-XBP 2.0 CFT mode, the target host to use for the command ID job step when the command ID option is used.
-cft_cmd_prefix	In pre-XBP 2.0 CFT mode, the prefix command required for the operating system of the target host.
-cft_abap	ABAP program to use for the command ID job step.
-cmdid	Identifier used to identify the unit of work represented by a USAP command and the associated SAP job.
-restart	Specification for whether or not this execution of Universal Connector is a restart of a previous client fault tolerant Universal Connector command.

Restart of a Command ID Job

In the following examples, Universal Connector is requested to restart command ID job E8E8E80001. Universal Connector will first parse the **jobdef** file to determine the jobname, and then scan the SAP system for a matching command ID job.

If a matching command ID job is found, Universal Connector reconnects to that job and satisfies the command line requirements. In this case, that means that if the job has not yet been started, it will be started, Universal Connector will wait for the job to complete (if it has not already), and the output will be returned.

If no matching command ID job is found, Universal Connector will terminate with an error code. Appropriate informational messages will be printed to standard error.

Pre-XBP 2.0 CFT Mode

```
usap -userid bob -pwd secret -submit jobdef -start -wait -cmdid E8E8E80001 -
restart yes
     -cft_secure_cft no
```

Secure CFT Mode

```
usap -userid bob -pwd secret -submit jobdef -start -wait -cmdid E8E8E80001 -
restart yes
     -cft_secure_cft yes
```

Command Line Options

Command line options used in these examples are:

Command Options	Description
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-submit	Defines a job to the SAP system.
-start	Starts the newly defined job.
-wait	Wait for the SAP job to complete processing.
-cmdid	Identifier used to identify the unit of work represented by a USAP command and the associated SAP job.
-restart	Specification for whether or not this execution of Universal Connector is a restart of a previous client fault tolerant Universal Connector command.
-cft_secure_cft	Mode of client fault tolerance to be used for the command invocation.

Run a Command ID Job Using Restart AUTO

In the following examples, Universal Connector will first scan the SAP system to determine if a matching command ID job exists.

If no matching command ID job is found, Universal Connector considers this to be the initial instance of this command ID job and defines the new command ID job to the SAP system. If a matching command ID job is found, Universal Connector reconnects with the existing SAP job.

After determining if the command ID job is initial or a restart, Universal Connector satisfies the command line requirements.

Pre-XBP 2.0 CFT Mode

```
usap -userid bob -pwd secret -submit jobdef -start -wait -cft_target_host
pwdf0196
     -cft_cmd_prefix "cmd /C" -cmdid E8E8E80001 -restart auto -
cft_secure_cft no
```

Secure CFT Mode

```
usap -userid bob -pwd secret -submit jobdef -start -wait -cmdid E8E8E80001 -
restart auto
     -cft_secure_cft yes
```

Command Line Options

Command line options used in these examples are:

Command Options	Description
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-submit	Defines a job to the SAP system.
-start	Starts the newly defined job.

-wait	Wait for the SAP job to complete processing.
-cft_secure_cft	Mode of client fault tolerance to be used for the command invocation.
-cft_target_host	In pre-XBP 2.0 CFT mode, the target host to use for the command ID job step when the command ID option is used.
-cft_cmd_prefix	In pre-XBP 2.0 CFT mode, the prefix command required for the operating system of the target host.
-cmdid	Identifier used to identify the unit of work represented by a USAP command and the associated SAP job.
-restart	Specification for whether or not this execution of Universal Connector is a restart of a previous client fault tolerant Universal Connector command.

Working With Pre-defined SAP Jobs

- Initial Run of a Command ID Job
  - Pre-XBP 2.0 CFT Mode
  - Secure CFT Mode
  - Command Line Options
- Restart of a Command ID Job
  - Pre-XBP 2.0 CFT Mode
  - Secure CFT Mode
  - Command Line Options
- Run a Command ID Job Using Restart AUTO
  - Pre-XBP 2.0 CFT Mode
  - Secure CFT Mode
  - Command Line Options

Initial Run of a Command ID Job

The following examples will submit, start, and wait for the command ID job defined in the pre-existing SAP job with job name 'JOB_A' and job ID 19561301. Because the RESTART option is set to **no**, Universal Connector will scan the SAP system to ensure that a command ID job with the same job name/command ID pair does not already exist on the system.

If a matching command ID job is found on the SAP system, Universal Connector will exit with an error code before performing the job submission.

Note that the Client Fault Tolerance Command ID Prefix is set up for a Windows host. In many user environments, the Client Fault Tolerance Command ID Prefix parameter can be specified in the configuration file and will never need to be specified on the command line. The same may be true for the Client Fault Tolerance Target Host parameter.

Pre-XBP 2.0 CFT Mode

```
usap -userid bob -pwd secret -submit -jobname JOB_A -jobid 19561301 -start -
wait
     -cft_target_host pwdf0196 -cft_cmd_prefix "cmd /C" -cmdid E8E8E80001 -
restart no
     -cft_secure_cft no
```

Note
The Client Fault Tolerance Command ID Prefix is set up for a Windows host. In many user environments, the Client Fault Tolerance Command ID Prefix parameter can be specified in the configuration file and will never need to be specified on the command line. The same may be true for the Client Fault Tolerance Target Host parameter.

Secure CFT Mode

```
usap -userid bob -pwd secret -submit -jobname JOB_A -jobid 19561301 -start -
wait
     -cft_secure_cft yes -cft_abap BTCTEST -cmdid E8E8E80001 -restart no
```

Note
In secure CFT mode, the **cft_secure_cft** and **cft_abap** parameters would most likely be specified in the Universal Connector configuration file.

Command Line Options

Command line options used in these examples are:

Command Options	Description
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-submit	Defines a job to the SAP system.
-jobname	Name of the SAP job.
-jobid	Job ID of the SAP job.
-start	Starts the newly defined job.
-wait	Wait for the SAP job to complete processing.
-cft_target_host	In pre-XBP 2.0 CFT mode, the target host to use for the command ID job step when the command ID option is used.
-cft_cmd_prefix	In pre-XBP 2.0 CFT mode, the prefix command required for the operating system of the target host.
-cmdid	Identifier used to identify the unit of work represented by a USAP command and the associated SAP job.
-restart	Specification for whether or not this execution of Universal Connector is a restart of a previous client fault tolerant Universal Connector command.
-cft_secure_cft	Mode of client fault tolerance to be used for the command invocation.

Restart of a Command ID Job

In the following example, Universal Connector is requested to restart command ID job E8E8E80001. Universal Connector will scan the SAP system for a matching command ID job.

If a matching command ID job is found, Universal Connector reconnects to that job and satisfies the command line requirements. In this case, that means that if the job has not yet been started, it will be started, Universal Connector will wait for the job to complete (if it hasn't already), and the output will be returned.

If no matching command ID job is found, Universal Connector will terminate with an error code. Appropriate informational messages will be printed to standard error.

Pre-XBP 2.0 CFT Mode

```
usap -userid bob -pwd secret -submit -jobname JOB_A -jobid 19561301 -start -
wait
     -cmdid E8E8E80001 -restart yes -cft_secure_cft no
```

Secure CFT Mode

```
usap -userid bob -pwd secret -submit -jobname JOB_A -jobid 19561301 -start -
wait
     -cmdid E8E8E80001 -restart yes -cft_secure_cft yes
```

Command Line Options

Command line options used in these examples are:

Command Options	Description
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-submit	Defines a job to the SAP system.

-jobname	Name of the SAP job.
-jobid	Job ID of the SAP job.
-start	Starts the newly defined job.
-wait	Wait for the SAP job to complete processing.
-cmdid	Identifier used to identify the unit of work represented by a USAP command and the associated SAP job.
-restart	Specification for whether or not this execution of Universal Connector is a restart of a previous client fault tolerant Universal Connector command.
-cft_secure_cft	Mode of client fault tolerance to be used for the command invocation.

Run a Command ID Job Using Restart AUTO

In the following examples, Universal Connector will first scan the SAP system to determine if a matching command ID job exists.

If no matching command ID job is found, Universal Connector considers this to be the initial instance of this command ID job and defines the new command ID job to the SAP system. If a matching command ID job is found, Universal Connector reconnects with the existing SAP job.

After determining if the command ID job is initial or a restart, Universal Connector satisfies the command line requirements.

Pre-XBP 2.0 CFT Mode

```
usap -userid bob -pwd secret -submit -jobname JOB_A -jobid 19561301 -start -
wait
     -cft_target_host pwdf0196 -cft_cmd_prefix "cmd /C" -cmdid E8E8E80001 -
restart auto
     -cft_secure_cft no
```

Secure CFT Mode

```
usap -userid bob -pwd secret -submit -jobname JOB_A -jobid 19561301 -start -
wait
     -cmdid E8E8E80001 -restart auto -cft_secure_cft yes
```

Command Line Options

Command line options used in these examples are:

Command Options	Description
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-submit	Defines a job to the SAP system.
-jobname	Name of the SAP job.
-jobid	Job ID of the SAP job.
-start	Starts the newly defined job.
-wait	Wait for the SAP job to complete processing.
-cft_target_host	In pre-XBP 2.0 CFT mode, the target host to use for the command ID job step when the command ID option is used.
-cft_cmd_prefix	In pre-XBP 2.0 CFT mode, the prefix command required for the operating system of the target host.
-cmdid	Identifier used to identify the unit of work represented by a USAP command and the associated SAP job.
-restart	Specification for whether or not this execution of Universal Connector is a restart of a previous client fault tolerant Universal Connector command.

-cft_secure_cft	Mode of client fault tolerance to be used for the command invocation.

Implementing Fault Tolerance - Examples

Implementing Fault Tolerance - Examples

- Implementing Manager Fault Tolerance for Windows

Implementing Manager Fault Tolerance for Windows

- Implementing Manager Fault Tolerance for Windows
  - Command Line Options
  - Components

Implementing Manager Fault Tolerance for Windows

The following figure illustrates how to activate manager fault tolerance. A unique command id is always required for manager fault tolerance.

```
ucmd -script script.file -host dallas -encryptedfile encrypted.file
     -managerft yes -cmdid uniquejobname -restart auto
```

The command is sent to a remote system, **dallas**, for execution. The output of the script is redirected back to the UCMD process. Additional command line options are read from the encrypted file, **`encrypted.file`**. Manager fault tolerance is turned on. A unique command ID, **`unique jobname`**, is coded to identify the process.

Restart is detected automatically. If an executing or pending command ID exists, a reconnect is performed. If not, the process is started as if new.

Command Line Options

The command line options used in this example are:

Options	Description
-script	File from which to read a script file. The script file is sent to the remote system for execution.
-host	Directs the command to a computer with a host name of **dallas**.
-encryptedfile	File from which to read an encrypted command options file.
-managerft	Specification for whether or not the manager fault tolerant feature is used.
-cmdid	Unique command ID associated the unit of work.
-restart	Specification for whether or not the manager is requesting restart.

Components

Universal Command Manager for Windows

Monitoring and Alerting

- Overview
- Monitoring of All Agents
  - Monitored Information
  - Polling
  - Alerts
- Querying for Job Status and Activity
- Additional Information

Overview

The Monitoring and Alerting feature of Universal Agent provides for the monitoring the status and activity of all Universal Automation Center Agents in an enterprise and the posting of alerts regarding the statuses.

Monitoring is provided through continuous Monitoring of All Agents or by Querying for Job Status and Activity of a specific Agent.

Monitoring of All Agents

Universal Agent provides for the continuous monitoring of all Agents in an enterprise through its Universal Enterprise Controller component.

Monitored Information

Universal Agent monitors for three types of information:

1. Alerts for all Agents and SAP systems being monitored
2. Jobs (active, completed, and failed) for all Agents being monitored
3. Systems (Agents and SAP systems) being monitored

This information can be viewed via the I-Activity Monitor UEC client application.

Polling

Universal Agent periodically polls each Agent and SAP system in an enterprise in order to retrieve its status information.

It determines whether or not a change in status of the Agent or SAP system has occurred since the last poll. If the status has changed, it sends this information to the I-Activity Monitor.

Alerts

Universal Agent sends out alerts to any connected Agent-monitoring applications whenever:

- Agent is unreachable.
- Agent is not responding.
- Agent component enters an orphaned or disconnected state.

These alerts are posted to the:

- Event Log (when running under Windows)
- Console (when running under z/OS)

Automation tools can be used in conjunction with these messages to perform operations based on agent failures.

Querying for Job Status and Activity

Universal Agent has the ability to query any specific Universal Broker in an enterprise for Broker-related, and active component-related, activity via the Universal Query utility.

Universal Query returns information for a Universal Broker that is installed on the host, as specified by configuration options on the command line or in a configuration file. Information regarding the components managed by a particular Universal Broker also can be requested.

Universal Query registers with a locally running Universal Broker. Consequentially, a Universal Broker must be running in order for a Universal Query to execute.

Additional Information

The following pages provide additional detailed information for Monitoring and Alerting:

- Monitoring and Alerting - Examples

Monitoring and Alerting - Examples

Examples

- Universal Query - Output
- Universal Query - z/OS
- Universal Query - UNIX and Windows
- Universal Query - IBM i

Note

The IBM i example references the IBM i command by its untagged name. If you are using commands with tagged names to run Universal Query, substitute the tagged name for this untagged name. (For information on tagged names, see UCHGRLS (Change Release Tag) Program.)

Universal Query - zOS

Universal Query for z/OS

The Universal Query utility is used to list all active components on a remote server.

The output will be written to the **SYSPRINT** DD statement.

```
//S1 EXEC UQRYPRC
//SYSIN    DD  *
-host dallas
/*
```

All active component information for server **dallas** will be written to DD statement **SYSOUT**.

SYSIN Option

The SYSIN option used in this example is:

Option	Description
-host	Directs the command to a computer with a host name of **dallas**.

Components

Universal Query

Universal Query - UNIX and Windows

Universal Query for UNIX and Windows

The Universal Query utility is used to list all active components on a remote server.

The output will be written to stdout.

```
uquery -host localhost
```

All active component information for the **localhost** server will be written to stdout.

Command Line Option

The command line option used in this example is:

Option	Description
-host	Directs the command to the **localhost**.

Components

Universal Query

Universal Query - IBM i

Universal Query for IBM i

The Universal Query utility is used to list all active components on a remote server.

The output will be written to stdout.

```
STRUQR HOST(localhost) PORT(4990)
```

This command provides active component information for the **localhost** server listening on port 4990 will be written to stdout.

```
STRUQR HOST(fortworth)
```

This command provides active component information from the **fortworth** server listening on the default port 7887.

Command Line Options

The command line options used in these examples are:

Option	Description
HOST	Directs the command to the **localhost**.
PORT	TCP port on the remote server.

Components

Universal Query

Universal Query - Output

Universal Query - Output

The following figure illustrates an example of the output generated by the execution of the Universal Query utility.

This sample output is from the execution of Universal Query to host **`dallas.domain.com`** using a NORMAL report.

```
  Universal Query Report

for
                        Thu 06 Sep 2018 05:54:00 PM
EDT

            host: 10.20.30.40  port: 7887  ping: NO  report:
NORMAL

                  Ubroker Host Name.....:
                  Ubroker IP Address....: *
                  Ubroker Host Port.....: 7887
                  Ubroker Description...: Universal Broker
                  Ubroker Version.......: 6.5.0 Level 0 Release Build 108
                  Ubroker Service.......: ubroker
                  Ubroker Status........: Active
                  Ubroker Managed.......: NO
                  Ubroker Start Time....: 03:35:52 PM
                  Ubroker Start Date....: 09/06/2018
                  UAG Netname(s)........: LX3RH7X64



               Component ID.............: 1121367481
               Component Name...........: ucmd
               Component Description....: Universal Command Server
               Component Version........: 6.5.0 Level 0 Release Build 108
               Component Type...........: ucmd
               Component Process ID.....: 773
               Component Start Time.....: 05:53:39 PM
```

```
Component Start Date......: 09/06/2018
Component Command ID......: sleep 60
Component State...........: REGISTERED
Component MGR UID.........: ucuser
Component MGR Work ID.....: PID12890
Component MGR Host Name...: dallas.domain.com
Component MGR IP Address..: 10.20.30.34
Component MGR Port........: 49082
Component Comm State......: ESTABLISHED
Component Comm State Time.: 05:53:41 PM
Component Comm State Date.: 09/06/2018
Component MGR Restartable.: NO
Component Comment.........: Sleep for 60 secs on dallas
```

Components

Universal Query

Messaging and Auditing

Overview

All Universal Agent components have the same message facilities. Messages — in this context — are text messages written to a console, file, or system log that:

1. Document the actions taken by a program.
2. Inform users of error conditions encountered by a program.

These pages describe the message and audit facilities that are common to all Universal Agent components. (See the individual Universal Agent component documentation for detailed technical information.)

Detailed Information

The following pages provide detailed information for Messaging and Auditing::

- Messaging
- Auditing
- Creating Write-to-Operator Messages - Examples

Messaging

- Message Types
- Message ID
- Message Levels
- Message Destinations
  - z/OS Message Destinations
  - Windows Message Destinations
  - UNIX Message Destinations
  - IBM i Message Destinations
  - HP NonStop Message Destinations

Message Types

There are six types (or severity levels) of Universal Agent messages. (The severity level is based on the type of information provided by those messages.)

Audit	Document the configuration options used by the program's execution and resource allocation details. They provide complete description of the program execution for auditing and problem resolution.
Informational	Document the actions being taken by a program. They help determine the current stage of processing for a program. They also document statistics about data processed.
Warning	Document unexpected behavior that may cause or indicate a problem.

Error	Document program errors. They provide diagnostic data to help identify the cause of the problem.
Diagnostic	Document diagnostic information for problem resolution.
Alert	Document a notification that a communications issue, which does not disrupt the program or require action, has occurred.

The MESSAGE_LEVEL configuration option in each Universal Agent component lets you specify which messages are written (see #Message Levels, below).

For a description of all Universal Agent messages, see Messages and Codes.

Message ID

Each message is prefixed with a message ID that identifies the message.

The message ID format is pppnnnnl, where:

- ppp is the product category identifier:
  - UAG (Universal Automation Center Agent Components)
  - UNV (Universal Components)
- nnnn is the message number.
- l is the message type (severity level):
  - A (Audit)
  - I (Informational)
  - W (Warning)
  - E (Error)
  - D (Diagnostic)
  - T (alerT)

Message Levels

Each Universal Agent component includes a MESSAGE_LEVEL configuration option that lets you select which types (severity levels) of messages are to be written.

- *Audit* specifies that all audit, informational, warning, and error messages are to be written.
- *Informational* specifies that all informational, warning, and error messages are to be written.
- *Warning* specifies that all warning and error messages are to be written.
- *Error* specifies that all error messages are to be written.
- *Trace* specifies that a trace file is created, to which data used for program analysis will be written. The trace file name and location are component-dependent (see the appropriate Universal Agent component documentation for details).
  (Trace should be used only at the request of Stonebranch, Inc. Customer Support.)

> **Note**
> Diagnostic and Alert messages always are written, regardless of the level selected in the MESSAGE_LEVEL option.

Message Destinations

The location to which messages are written is the message destination.

Some Universal Agent components have a MESSAGE_DESTINATION configuration option that specifies the message destination. If a program is used only from the command line or batch job, it may have only one message destination, such as standard error.

Valid message destination values depend on the host operating system.

z/OS Message Destinations

Universal Agent on z/OS run as batch jobs or started tasks. Batch jobs do not provide the MESSAGE_DESTINATION option. All messages are written to the **SYSOUT** ddname.

Started task message destinations are listed in the following table.

Destination	Description
LOGFILE	Messages are written to ddname UNVLOG. All messages written to log files include a date and time stamp and the program's USS process ID.

SYSTEM	Messages are written to the console log as WTO messages.

Windows Message Destinations

Message destinations are listed in the following table.

Destination	Description
STDERR	Messages are written to standard error. This destination is most useful for console commands.
LOGFILE	Messages are written to a log file. Not all programs provide this destination. Log files are written to product specific log directories, which can be modified with the LOG_DIRECTORY option. All messages written to log files include a date and time stamp and the program's process ID.
SYSTEM	Messages are written to the Windows Application Event Log.

UNIX Message Destinations

Message destinations are listed in the following table.

Destination	Description
STDERR	Messages are written to standard error. This destination is most useful for console commands.
LOGFILE	Messages are written to a log file. Not all programs provide this destination. The recommended directory for log files is **`/var/opt/universal/log`**. This can be changed with the LOG_DIRECTORY option. All messages written to log files include a date and time stamp and the program's process ID.
SYSTEM	Messages are written to the **syslog** daemon. Not all programs provide this destination. Universal programs that execute as daemons write to the **syslog**'s daemon facility. All messages include the programs process ID. If an error occurs writing to the **syslog**, the message is written to the system console.

IBM i Message Destinations

Message destinations are listed in the following table.

Destination	Description
STDERR	Messages are written to standard error. A batch job's standard error file is allocated to the print file QPRINT.
LOGFILE	Messages are written to the job's job log.
SYSTEM	Messages are written to the system operator message queue QSYSOPR.

HP NonStop Message Destinations

Message destinations are listed in the following table.

Destination	Description
STDERR	Messages are written to standard error.
LOGFILE	Messages are written to a log file. Not all programs provide this destination. Log files are written the **$SYSTEM.UNVLOG** subvolume. All messages written to log files include a date and time stamp and the program's process ID.

Auditing

Auditing

Within Universal Agent, an event is the occurrence of some action or condition at a particular location in the computer network and at a particular time at that location. There are a number of different types of events, such as the start of a Universal Agent component, a user authentication failure, or a file transfer completion.

The Universal Event Subsystem (UES) provides the means by which Universal Agent components generate data about those events and, in a single repository, have those events recorded. This collection of recorded events (that is, the event records) is maintained in the UES database and archived to external storage. It represent the work and activity of all distributed workload managed by Universal Agent components.

Universal Agent consists of a set of components distributed across a computer network. The components work together to perform some unit of work. The components that are working together have an association that must be maintained in the event data. For that reason, UES event records not only include information about the event, but also information about associations between the components reporting the events.

Universal Enterprise Controller (UEC) maintains a central UES database for all event data within its domain of responsibility. The UES database contains all UES event records collected by UEC from Universal Broker components that are defined to it. The UES database provides medium-term persistent storage for the UES events. Periodically, the UES database events must be exported to long-term storage in order to maintain a historical record of events. If the export is not performed periodically, the UES database will continue to grow and eventually exhaust all disk space available to it.

Examples of components and their associations are:

- Universal Command Manager is associated with a remote Universal Command Server, and the Universal Command Server is associated with the job process it has started on behalf of the Universal Command Manager.
- Universal Data Mover Manager is associated with a remote Universal Data Mover Server, and the Universal Data Mover Server is associated with a file being transferred on behalf of the Universal Data Mover Manager.

The components and their associations partly define the Universal Agent architecture. This section provides the necessary understanding of the Universal Agent architecture as presented by the UES event data.

Creating Write-to-Operator Messages - Examples

Creating Write-to-Operator Messages - Examples

- Issue WTO Message to z/OS Console
- Issue WTO Message to z/OS Console and Wait for Reply

Issue WTO Message to zOS Console

Issue WTO Message to z/OS Console

The following illustrates the issuing of a WTO message to the z/OS console.

No reply is required.

```
uwto -msg "This message is written to the Console"
```

The message text "**This message is written to the Console**" will be written to the default z/OS consoles.

SYSIN Options

The SYSIN option used in this example is:

Option	Description
-msg	Text to write to the z/OS operator console. The text is written as a single-line WTO or WTOR message.

Components

Universal Write-to-Operator

Issue WTO Message to zOS Console and Wait for Reply

Issue WTO Message to z/OS Console and Wait for Reply

The following illustrates the issuing of a WTOR message to the z/OS console.

A reply is required.

```
uwto -msg "This message is written to the Console" -reply yes -timeout 120
```

The message text "**This message is written to the Console**" will be written to the default z/OS consoles.

The process will wait 120 seconds for a required reply. If a reply is not received within this time, the WTOR message is deleted and Universal WTO ends with exit code 2. The reply length is limited to 119 characters. The reply is written to UWTO's standard output file.

> **Note**
> A valid operator reply to a WTOR message can be zero characters. In this case, nothing is written to stdout.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-msg	Text to write to the z/OS operator console. The text is written as a single-line WTO or WTOR message.
-reply	Directs Universal WTO to issue a WTOR message and wait for an operator reply to the message.
-timeout	Number of seconds to wait for a WTOR operator reply. If a reply is not received within this time, the WTOR message is deleted and UWTO ends with exit code 2. Default is 0 (wait indefinitely).

Components

Universal Write-to-Operator

Message Translation

- Overview
- Usage
  - Translation Table
  - Matching Algorithm
- Additional Information

Overview

Universal Agent component error messages are translated - by the Universal Message Translator (UMET) utility - into return (exit) codes based on a user-defined translation table.

Every command ends with a return code that indicates the success or failure of the command execution. Typically, a return code of 0 indicates success; all other codes indicate failure.

However, a small number of commands do not set their return code under failure conditions; instead, they issue error messages. Based on the user-defined translation table, Universal Message Translator translates these error messages into return codes.

Usage

Universal Message Translator requires two input files:

1. Message Input file (user-specified or standard input) containing the error messages that are to be translated into a return codes.
2. Translation Table file containing the user-defined translation table that controls the error message-to-return code translation process.

To perform a translation, Universal Message Translator:

1. Reads the messages in the input file.
2. Matches each line against the translation table entries.
3. Exits with an return code from the best match in the translation table.

If no match is found, Universal Message Translator ends with return code 0.

Universal Message Translator performs operations specified by the configuration options. This section describes each option and their syntax.

Translation Table

The translation table specifies:

- Text to search for.
- Return code associated with the text.
- Precedence when multiple matches are found.

Translation Table Format

The translation table consists of one or more lines.

Each line is either:

- Comment line (# in column one)
- Blank line (ignored)
- Translation table entry

Translation table entries consist of two fields separated by spaces or tabs. An entry cannot be continued onto multiple lines.

Translation Table Fields

Field	Description
Message Mask	Selects which messages to match in the input file. The mask must be enclosed in double (**"**) quotation marks. Mask characters include the asterisks (*****) and the question mark (**?**). The asterisk matches 0 or more characters and the question mark matches one character. If an asterisk, question mark, or quotation mark is required in the message text, it must be preceded with a back slash (****). If a back slash is required in the message text, it must be preceded by another back slash.
Exit Code	Specifies an integer value that UMET exits with if this entry is the resulting match. The exit code is in the range of -99999 to 99999.

Matching Algorithm

The input file is read line by line. For each line, the line is compared to each entry in the translation table. All the matching entries are saved.

After the entire input file is read, the matched entries from the translation table are sorted in ascending order by their line number in the translation table. The first entry in this sorted list is the resulting translation table entry. The exit code from the resulting translation table entry is used as the return code of UMET. If no matching entry is found, UMET exits with 0.

IBM i

The resulting return code from the translation process is converted into an IBM i escape message.

The escape message ID and message severity depend on the return code value as identified in the following table.

Return Code	Message ID	Message Severity
1 - 10	UNV0344	10
11 - 20	UNV0345	20
21 - 30	UNV0346	30
31 and higher	UNV0347	40

Additional Information

The following pages provide additional detailed information for Message Translation:

- Message Translation - Examples

Message Translation - Examples

- Examples

Examples

- Translating Error Messages
- Execute Universal Message Translator from zOS
- Execute UMET from zOS Manager (with Table on Remote Server)
- Execute UMET from zOS Manager (with Table on zOS)
- Execute Universal Message Translator from Windows
- Execute Universal Message Translator from UNIX
- Execute Universal Message Translator from IBM i

Note
The IBM i example references the IBM i command by its untagged name. If you are using commands with tagged names to run Universal Message Translator, substitute the tagged name for this untagged name. (For information on tagged names, see UCHGRLS (Change Release Tag) Program.)

Translating Error Messages

- Example 1
- Example 2
  - Components

Note
These examples are not specific to any particular operating system.

Example 1

In this example, a command generates the following **stderr** file.

```
Error opening rc file /etc/arc.rc
No rc file opened.
Ending due to error.
```

From the contents of the message file, we can see that the program failed to open a resource configuration file.

Either of the following translation tables could match error messages in the message file. Message masks should be general enough to match a set of error messages.

Translation Table 1

```
# UMET Translation Table 1
#
# Message Mask                Exit Code
# ------------                ---------
  "*error*"                   8
```

Translation Table 1 will result in a match if any input line contains the word **error**. The resulting exit code will be *8* if a match occurs.

Translation Table 2

```
# UMET Translation Table 2
#
# Message Mask                Exit Code
# ------------                ---------
 "Ending due to error."       8
```

Translation Table 2 will result in a match only if the exact message text **"Ending due to error."** appears as a line in the input file. This is less general, but may be sufficient for this command.

Example 2

(This example continues from Example 1.)

In this example, the command now generates the following **stderr** file.

```
Error opening rc file /etc/arc.rc
Processing rc file /usr/etc/arc.rc
Ending successfully
```

From the contents of the message file, we can see that the program failed to open a resource configuration file **/etc/arc.rc**, but successfully opened file **/usr/etc/arc.rc**.

Translation table

The following translation table is one of many that could match error messages in the message file.

```
# UMET Translation Table 1
#
# Message Mask                Exit Code
# ------------                ---------
  "Ending due to error."      8
  "Processing rc file *"      0
  "Error opening rc file *"   8
```

Translation Table 1 contains three entries:

- First entry matches against a specific error message that always indicates an error if present.
- Second and third entries match messages produced by resource configuration file processing.

Components

Universal Message Translator

Execute Universal Message Translator from zOS

- Execute Universal Message Translator from z/OS
  - PARM Options
  - Components

Execute Universal Message Translator from z/OS

The following figure illustrates the execution of Universal Message Translator from z/OS.

```
//S1 EXEC PGM=UMET,PARM='-table tabledd -level verbose'
//STEPLIB  DD  DISP=SHR,DSN=hlq.UNV.SUNVLOAD
//SYSPRINT DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
//TABLEDD  DD *
"*ERROR*"       8
"*WARN*"        4
"*ERROR*"       7
/*
//SYSIN    DD *
THIS IS AN ERROR MESSAGE RESULTING IN RETURN CODE 8.
/*
```

The -table option points to the DD statement **TABLEDD**, which defines the return codes to end this process based on matching text. The first column defines the text to match; the second defines the return code to set if the matching text exists in the **SYSIN** DD.

The -level option turns on messaging. All messages will be written to **SYSPRINT**. The **SYSIN** DD statement points to the text file to be interrogated.

PARM Options

The PARM options used in this example are:

Option	Description
-table	Translation table file name.
-level	Level of messages that will be displayed.

Components

Universal Message Translator

Execute UMET from zOS Manager (with Table on Remote Server)

- Execute Universal Message Translator from z/OS Manager (in a Script with Table Housed on Remote Server)
  - Script Options
  - SYSIN Options
  - Components

Execute Universal Message Translator from z/OS Manager (in a Script with Table Housed on Remote Server)

The following figure illustrates the execution of Universal Message Translator from a z/OS Universal Command Manager.

```
//S1 EXEC UCMDPRC
//SCRIPTDD DD *
/opt/universal/ucmdsrv-2.2.0/bin/umet -file /home/log.file -table\
/home/umet.table -level verbose
/*
//SYSIN    DD  *
-host dallas
-script SCRIPTDD
-userid joe
-pwd abcdefg
/*
```

Universal Message Translator is executed in order to interrogate a log file and set the return code based on the translation table.

Since the command spans two lines, the native operating system continuation character must be used:

- \ for UNIX
- ¬ for Windows

The full path to the Universal Message Translator executable must be specified for UNIX if the path is not part of the user's profile.

Script Options

The script options used in this example are:

Option	Description
-file	Input message file name. If the option is not specified, UMET reads its input from **stdin**.
-table	Translation table file name.
-level	Level of messages that will be displayed.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-host	Host name or IP address of the remote system on which to execute the script.
-script	ddname from which to read a script file. The script file is sent to the remote system by UCMD Manager for execution

-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

Components

Universal Command Manager for z/OS

Universal Message Translator

Execute UMET from zOS Manager (with Table on zOS)

- Execute Universal Message Translator from z/OS Manager (in a Script with Table Housed on z/OS)
  - Script Options
  - SYSIN Options
  - Components

Execute Universal Message Translator from z/OS Manager (in a Script with Table Housed on z/OS)

The following figure illustrates the execution of Universal Message Translator from a z/OS Universal Command Manager.

```
//S1 EXEC UCMDPRC
//UNVIN    DD DISP=SHR,DSN=hlq.umet.table
//SCRIPTDD DD *
UCOPY > c:\temp\umet.table
umet -table c:\temp\umet.table -file c:\temp\bkup.log -level
verbose
/*
//SYSIN   DD  *
-host dallas
-script SCRIPTDD
-userid joe
-pwd abcdefg
/*
```

The message table is stored and maintained on z/OS and copied down to the server upon execution. The -table option points to the table of defined return codes based on text. The -file option points to the text file to be interrogated.

The first command copies the messages table from the **UNVIN** DD of the manager process to a server file named **`c:\temp\umet.table`**. The UMET program then is executed to interrogate the log file and set the return code based on the translation table.

Script Options

The script options used in this example are:

Option	Description
-table	Translation table file name.
-file	Input message file name. If the option is not specified, UMET reads its input from **stdin**.
-level	Level of messages that will be displayed.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-host	Host name or IP address of the remote system on which to execute the script.
-script	ddname from which to read a script file. The script file is sent to the remote system by UCMD Manager for execution
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

Components

[Universal Command Manager for z/OS

Universal Message Translator

Execute Universal Message Translator from Windows

- Execute Universal Message Translator from Windows
  - Command Line Options
  - Components

Execute Universal Message Translator from Windows

The following figure illustrates the execution of Universal Message Translator from Windows.

```
-table c:\umettable.txt -file c:\umetfile.txt -level verbose
```

The -table option points to the file that defines the return codes with which to end this process, based on matching text.

The first column of the file defines the text to match; the second defines the exit code to set if the matching text exists in the file defined by the -file option.

The -level option turns on messaging. All messages will be written to **stdout**.

Command Line Options

The command line options used in this example are:

Option	Description
-table	Translation table file name.
-level	Level of messages that will be displayed.
-file	Input message file name. If the option is not specified, UMET reads its input from **stdin**.

Components

Universal Message Translator

Execute Universal Message Translator from UNIX

- Execute Universal Message Translator from UNIX
  - Command Line Options
  - Components

Execute Universal Message Translator from UNIX

The following figure illustrates the execution of Universal Message Translator from UNIX.

Although the command is shown on two lines, it should be entered on one line at the command prompt or within a script, or it can be continued within the script with the UNIX continuation character ****.

The -table option points to the file, which defines the return codes with which to end this process, based on matching text.

```
/opt/universal/ucmdsrv-2.2.0/bin/umet -table /tmp/umettable.txt -file /tmp
/umetfile.txt -level verbose
```

The first column of the file defines the text to match; the second defines the return code to set if the matching text exists in the file defined by the -file option.

The -level option turns on messaging. All messages will be written to **stdout**.

Command Line Options

The command line options used in this example are:

Option	Description
-table	Translation table file name.
-level	Level of messages that will be displayed.
-file	Input message file name. If the option is not specified, UMET reads its input from **stdin**.

Components

Universal Message Translator

Execute Universal Message Translator from IBM i

- Execute Universal Message Translator from IBM i
  - Command Line Options
  - Components

Execute Universal Message Translator from IBM i

The following example illustrates the execution of Universal Message Translator from IBM i.

```
STRUME MSGFILE(input_file) MSGMBR(member) TBL(table_file) TBLMBR(member)
MSGLEVEL(*VERBOSE)
```

The TBL [TBLMBR] option points to the file, which defines the exit codes with which to end this process, based on matching text. The first column of the file defines the text to match; the second defines the return code to set if the matching text exists in the file defined by the MSGFILE [MSGMBR] option.

Diagnostic message UNV0383 and Informational message CPF9815 are issued if an error occurs during execution of the STRUME command. All other informational messages will be written to STDOUT. To avoid messages written to stdout, either allow MSGLEVEL to default to ***warn** or specify MSGLEVEL as ***error**.

Command Line Options

The command line options used in this example are:

Option	Description
TBL [TBLMBR]	Translation table file name.
MSGLEVEL	Level of messages that will be displayed.
MSGFILE [MSGMBR]	Input message file name. If the option is not specified, UMET reads its input from **stdin**, which is allocated to the terminal for interactive jobs and to QINLINE for non-interactive jobs.

Components

Universal Message Translator

Network Data Transmission for Universal Agent

- Introduction
- Network Protocols
- Detailed Information

Introduction

Distributed systems, such as Universal Agent, communicate over data networks. All Universal Agent components communicate using the TCP/IP protocol; they do not use the UDP protocol for any product data communication over a network.

Network Protocols

Universal Agent can utilize either of two network protocols:

- **Secure Socket Layer/Transport Layer Security Protocol**
  Secure Socket Layer/ Transport Layer Security provides the highest level of security available. SSL/TLS is a widely used and accepted network protocol for distributed software applications that are required to address all aspects of secure data transfer on private and public networks. All Universal Agent components (version 3.x and later) use SSL/TLS.
- **Universal V2 Protocol**
  Universal V2 (version 2) legacy protocol, **UNVv2**, is provided for backward compatibility with Universal Agent (formerly Universal Products) versions earlier than 3.x, and when the SSL/TLS protocol resource utilization is considered too high. To ensure backward compatibility, this protocol is still supported by version 3.x components.

In addition to the network protocol used to transmit data, Universal Agent components use an application-layer protocol, Universal Application Protocol, to exchange data messages.

Detailed Information

The following pages provide detailed information for Network Data Transmission:

- SSL/TLS Protocol
- Universal V2 Protocol
- Universal Application Protocol
- Network Data Transmission Tuning
- Network Data Transmission Configurable Options

SSL/TLS Protocol

- Overview
- Data Privacy and Integrity
  - Encryption Algorithms
  - Message Digest Algorithms
  - Supported SSL/TLS 1.2 Cipher Suites
  - Supported SSL/TLS 1.3 Cipher Suites
- Peer Authentication

Overview

Universal Agent implements the SSL/TLS (Secure Socket Layer/Transport Layer Security) protocol using the OpenSSL library or the IBM z/OS System SSL library, available on the z/OS operating system. The most recent SSL standard is version3. A subsequent version was produced, changing the name to Transport Layer Security version 1 (**TLSv1**). **TLSv1** is the actual protocol used by Universal Agent. **TLSv1** is often referred to simply as SSL. The term SSL/TLS is used throughout the rest of this documentation to mean **TLSv1**, unless otherwise noted.

Subsequent SSL/TLS protocol versions have been produced. This includes Transport Layer Security version 1.2 (**TLSv1.2**) and 1.3 (**TLSv1.3**). **TLSv1.2** is the current default Agent protocol used. However, the Agent still supports **TLSv1** and will negotiate down to **TLSv1** to communicate with older versions of the Agent. For enhanced security constraints, the Agent provides ways to restrict the SSL/TLS protocol used to a minimum level and impose more strict rules around the SSL/TLS protocol that is allowed.

The SSL/TLS protocol addresses the major challenges of communicating securely over a potentially insecure data network. This page discusses the issue of data privacy and integrity, and peer authentication.

Data Privacy and Integrity

People with sufficient technical knowledge and access to network resources can watch or capture data transmitting across the network. What they do with the data is up to them.

Data sent over the network that should remain private must be encrypted in a manner that unauthorized persons cannot determine what the original data contained regardless of their level of expertise, access to network resources, amount of data captured, and amount of time they have. The only party that should be able to read the data is the intended recipient.

As data is transmitted over the network, it passes through media and hardware of unknown quality that may erroneously change bits of data without warning. Additionally, although data may be encrypted, there is nothing stopping a malicious person from changing the data while it is transmitted over the network. The changed data may or may not be detected by the recipient depending on what changed and how it is processed. It may be accepted as valid data, but the information it represents is now erroneous

Data integrity must be protected from errors in transmission and malicious users. Data integrity checks insures that what was sent is exactly what is received by the recipient. Without integrity checks, there is no guarantee.

Encryption Algorithms

Encryption algorithms are used to encrypt data into an unreadable format. The encryption process is computationally expensive. There are a variety of encryption algorithms, some of which perform better than others. Some algorithms offer a higher level of security than others. Typically, the higher level of security requires more computational resources.

Message Digest Algorithms

Message digest algorithms are used to produce a Message Authentication Code (MAC) that uniquely identifies a block of data. The sender computes a MAC for the data being sent based on a shared secret key the sender and receiver hold. The sender sends the data and the MAC to the receiver. The receiver computes a new MAC for the received data based on the shared secret key. If the two MACs are the same, data integrity is maintained, else the data is rejected as it has been modified.

Message digest algorithms are often referred to as MACs and can be used synonymously in most contexts.

Supported SSL/TLS 1.2 Cipher Suites

The SSL standard defines a set of encryption and message digest algorithms, referred to as cipher suites, that insure data privacy and data integrity. Cipher suites pair encryption algorithms with appropriate message digest algorithms. The two algorithms cannot be specified individually.

Universal Agent supports a subset of the complete SSL cipher suites defined by the standard. The cipher suite name is formatted as an encryption algorithm abbreviation followed by the message digest algorithm abbreviation.

New Installations

For new installations of Universal Agent 6.3.0.1 or later, the following table identifies the supported SSL/TLS 1.2 (and prior) cipher suites and the default order in which they are selected (see Selecting an SSL Cipher Suite, below).

Cipher Suite Name	Description
AES256-GCM-SHA384	256-bit AES encryption in Galois Counter Mode, SHA-2 384-bit message digest.
AES256-SHA	256-bit AES encryption with SHA-1 message digest.
AES128-GCM-SHA256	128-bit AES encryption in Galois Counter Mode, SHA-2 256-bit message digest.
AES128-SHA	128-bit AES encryption with SHA-1 message digest.
RC4-SHA	128-bit RC4 encryption with SHA-1 message digest.
RC4-MD5	128-bit RC4 encryption with MD5 message digest.
DES-CBC3-SHA	128-bit Triple-DES encryption with SHA-1 message digest.
DES-CBC-SHA	128-bit DES encryption with SHA-1 message digest. **Note** As of Universal Agent 6.7.0.0, DES-CBC-SHA is supported only on HP-UX. Additionally, any 7.2.x Agents on HP-UX that accept connections from, or attempt connections to, 7.2.x Agents on other platforms must be configured with at least one currently supported cipher suite besides DES-CBC-SHA. Therefore, those HP-UX Agents cannot be configured only with DES-CBC-SHA in their list of cipher suites.
NULL-SHA256	No encryption and SHA-2 256-bit message digest.

NULL-SHA	No encryption and SHA-1 message digest.
NULL-MD5	No encryption and MD5 message digest.
NULL-NULL	No encryption, no data authentication, SSL is not used; instead, Universal V2 Protocol (**UNVv2**) is used.

The NULL-* ciphers are valid for most components, with these exceptions:

- Universal Broker does not offer NULL-* options for its ciphers list, but it does accept NULL-NULL when no encryption is desired.
- UCTL Server and UEM Server do not allow NULL-* ciphers to be selected for their control sessions.
- UDM Manager ignores the NULL-NULL cipher suite.

Starting with 7.4.0.0, the following Diffie-Hellman Ciphers are supported for **TSLv1.2 only**.

Cipher Suite Name	Description
ECDHE-RSA-AES256-GCM-SHA384	Ephemeral Elliptic Curve Diffie-Hellman Key Exchange, RSA authentication, 256-bit AES encryption in Galois Counter Mode, SHA-2 384-bit message digest.
ECDHE-ECDSA-AES256-GCM-SHA384	Ephemeral Elliptic Curve Diffie-Hellman Key Exchange, ECDSA authentication, 256-bit AES encryption in Galois Counter Mode, SHA-2 384-bit message digest.
ECDHE-RSA-AES128-GCM-SHA256	Ephemeral Elliptic Curve Diffie-Hellman Key Exchange, RSA authentication, 128-bit AES encryption in Galois Counter Mode, SHA-2 256-bit message digest.
ECDHE-ECDSA-AES128-GCM-SHA256	Ephemeral Elliptic Curve Diffie-Hellman Key Exchange, ECDSA authentication, 128-bit AES encryption in Galois Counter Mode, SHA-2 256-bit message digest.

Note
The ECDHE-ECDSA-AES256-GCM-SHA384 and ECDHE-ECDSA-AES128-GCM-SHA256 ciphers require an ECDSA certificate; A RSA certificate will NOT work for these ciphers. See Creating an ECDSA Certificate.

Furthermore, **TLSv1.2** only supports the following Elliptic Curves:

- prime256v1 (same as secp256r1)
- secp384r1
- secp521r1

As such, the EC (Elliptic Curve) keys used to sign the ECDSA certificate must be generated using one of the curves in the list above.

Supported SSL/TLS 1.3 Cipher Suites

The following table identifies all SSL/TLS 1.3 cipher suites provided by Stonebranch, Inc.

The list is in default order, with the most preferred suite first and the least preferred suite last.

Cipher Suite	Description
TLS_AES_256_GCM_SHA384	256-bit AES encryption in Galois Counter Mode, SHA-2 384-bit message digest
TLS_CHACHA20_POLY1305_SHA256	256-bit CHACHA encryption with POLY1305 message authentication, SHA-2 256-bit message digest
TLS_AES_128_GCM_SHA256	128-bit AES encryption in Galois Counter Mode, SHA-2 256-bit message digest

Upgrade Installations

For upgrade installations to Universal Agent 6.3.0.1 or later, any configured SSL cipher suite values will remain as is. The following table identifies the supported SSL cipher suites for Agent 6.3.0.0 and earlier, and the default order in which they are selected (see SSL/TLS Protocol - COMPLETE#Selecting an SSL Cipher Suite, below).

Cipher	Description

Suite Name	
AES256-SHA	256-bit AES encryption with SHA-1 message digest
AES128-SHA	128-bit AES encryption with SHA-1 message digest
RC4_SHA	128-bit RC4 encryption with SHA-1 message digest
RC4_MD5	128-bit RC4 encryption with MD5 message digest
DES_CBC3_SHA	128-bit Triple-DES encryption with SHA-1 message digest
DES-CBC-SHA	128-bit DES encryption with SHA-1 message digest. **Note** As of Universal Agent 6.7.0.0, DES-CBC-SHA is supported only on HP-UX. Additionally, any 7.2.x Agents on HP-UX that accept connections from, or attempt connections to, 7.2.x Agents on other platforms must be configured with at least one currently supported cipher suite besides DES-CBC-SHA. Therefore, those HP-UX Agents cannot be configured only with DES-CBC-SHA in their list of cipher suites.

Universal Agent supports one additional cipher suite name that is not part of the SSL protocol. The NULL-NULL cipher suite turns SSL off completely and instead uses the Universal V2 Protocol (**UNVv2**).

We'll continue to offer NULL-* values (i.e., no encryption, message authentication only) at the end of any default lists that currently offer them.

SSL Cipher Suites to be Deprecated

RC4_* and DES_* SSL cipher suites will be deprecated in a future release of Universal Agent.

Selecting an SSL Cipher Suite

When two Universal Agent components (for example, a UEM Manager and a UEM Server) first connect, they perform an SSL handshake that negotiates the cipher suite to use for the session. The Manager presents a list of cipher suites (in descending order of preference) that it would like to use. This is compared against a list of ciphers that the Server supports. The first cipher suite in common is the one used for the session.

Lists of cipher suites are helpful where a distributed software solution may cross many organizational and application boundaries, each with its own security requirements. Instead of having to choose one cipher suite for all distributed components, the software components can be configured with their own list of acceptable cipher suites based on their local security requirements.

When a high level of security is required, the higher CPU consuming cipher suite is justified. When lower level of security is acceptable, a lower CPU consuming cipher suite can be used. As long as the Manager has both cipher suites in its list, it can negotiate either cipher suite with servers of different security levels.

Peer Authentication

When communicating with a party across a data network, how do you insure that the party you are communicating with (your peer) is who you believe? A common form of network attack is a malicious user representing themselves as another user or host.

Peer authentication insures that the peer is truly who they identify themselves as. Peer authentication applies to users, computer programs and hardware systems.

SSL uses X.509 certificates and public and private keys to identify an entity. An entity may be a person, a program, or a system. A complete description of X.509 certificates is beyond the scope of this documentation. X.509 Certificates provides an overview to help get the reader oriented to the concepts, terminology and benefits.

For additional details, the following web site is recommended:

http://www.faqs.org/rfcs/rfc3280.html

Universal V2 Protocol

Universal V2 Protocol

The Universal V2 protocol, **UNVv2**, is a proprietary protocol that securely and efficiently transports data across data networks. **UNVv2** was the only supported protocol in Universal Agent (formerly Universal Products) prior to version 3 and will be available in future versions.

UNVv2 addresses data privacy and integrity. It does not address peer authentication.

Data Privacy and Integrity

Data privacy is insured with data encryption algorithms. **UNVv2** utilizes 128-bit RC4 encryption for all data encryption.

Data integrity is insured with message digest algorithms. **UNVv2** utilizes 128-bit MD5 MACs for data integrity. **UNVv2** referred to data integrity as data authentication.

Encryption and integrity may be enabled and disabled on an individual bases.

Encryption keys are generated using a proprietary key agreement algorithm. A new key is created for each and every network session.

Universal Application Protocol

- Universal Application Protocol
- Low-Overhead
- Secure
- Extensible

Universal Application Protocol

Universal Agent components use an application-layer protocol to exchange data messages. The protocol has the following characteristics:

- Low-Overhead
- Secure
- Extensible
- Configurable Options

The following information refers to two categories of data transmitted by Universal Agent:

1. Control data (or messages) consists of messages generated by Universal Agent components in order to communicate with each other. The user of the product has no access to the control data itself.
2. Application data (or messages) consists of data that is transmitted as part of the requested work being executed. For example, standard input and output data of jobs Universal Command executes. The data is created by the job and read or written by Universal Command on behalf of the job.

Low-Overhead

The protocol is lightweight, in order to minimize its use of network bandwidth. The product provides application data compression options, which reduces the amount of network data even further.

There are two possible compression methods:

1. **ZLIB** method offers the highest compression ratios with highest CPU utilization.
2. **HASP** method offers the lowest compression ratios with lowest CPU utilization.

> **Note**
> Control data is not compressed. Compression options are available for application data only.

Secure

When used by Universal Agent Managers prior to version 3.x, and when communicating with Universal Agent Servers that force encryption on, the UNVv2 protocol is secure.

All control data exchanged between Universal Agent components are encrypted with a unique session key and contain a MAC. The encryption prevents anyone from analyzing the message data and attempting to circumvent product and customer policies. Each session uses a different encryption key to prevent "play back" types of network attacks, where messages captured from a previous session are replayed in a new session. This applies to both network protocols: SSL/TLS and UNVv2.

In versions prior to Universal Agent, the security features used in the control messages are not optional. They cannot be turned off. The security features are optional for application data sent over the network.

Starting with Universal Agent, the UNVv2 protocol is used only when SSL/TLS is disabled on the control session by specifying the NULL-NULL cipher suite. In this case, the UNVv2 encryption or MACs are not used for control messages.

As of Universal Agent, the SSL/TLS protocol must be used if data privacy and integrity is required for control messages. For this reason, UNVv2 should only be used when the resource utilization of SSL/TLS is considered too high and data privacy is not required. It is Stonebranch's recommendation that SSL/TLS should be used if at all possible to insure data privacy and data integrity.

Backward compatibility is still maintained with Universal Agent (formerly Universal Products) versions prior to 3.x such that encryption and MACs are still utilized for the control session.

Extensible

The message protocol used between the Universal Agent components is extensible. New message fields can be added with each new release without creating product component incompatibilities. This permits different component versions to communication with each other with no problems. This is a very important feature for distributed systems, since it is near impossible to upgrade hundreds of servers simultaneously.

New encryption and compression algorithms can be added in future releases without loosing backward compatibility with older releases. After a network connection is made, connection options are negotiated between the two Universal Agent programs. The options negotiated include which encryption and compression algorithms are used for the session. Only algorithms that both programs implement are chosen in the negotiation process. The negotiation process permits two different program versions to communicate.

Network Data Transmission Tuning

- Overview
- Bandwidth Delay Product
- TCP High Performance Extensions
- TCP Buffers
  - TCP Buffer Configuration

Overview

TCP/IP tuning is considered an advanced subject. The background information necessary for a complete understanding of the subject is beyond the scope of this document. If misconfigured, the product configuration options for data transmission tuning can have a negative impact on the network performance of product components as well as on the TCP/IP network.

Product components that support bulk data transfers benefit the most from the network tuning described in this section. For example, Universal Data Mover provides the ability to transfer very large files between systems, so it would be a candidate for network tuning.

The network tuning technique described in this section addresses the problem of transferring data over certain types of transmission links, primarily large bandwidth, high latency transmission links. Today's transmission links can exceed 1 Gbit/s with a round trip time of 50 ms or more. The default TCP/IP buffers are not suitable for optimized data transmission over such links.

Bandwidth Delay Product

The bandwidth delay product (BDP) measures the amount of data that a transmission link holds. The BDP is used to help tune specific TCP/IP configuration options.

The BDP is calculated as the product of the maximum bandwidth and the round trip time (RTT) of the transmission link. BDP is expressed in bytes. The maximum bandwidth of a link is limited by the slowest part, or bottleneck, of the network route. As an example, consider a network route that starts on a server with a 100 Mbit/s network interface card that is connected to a 1 Gbit/s network and ends on a server with a 1 Gbit/s network interface card. The maximum bandwidth is the slowest part of the route, which is the 100 Mbit/s network interface card. There is no reliable way to measure bandwidth in all cases. A knowledge of the network topology is required to know the slowest part of a network route. The RTT is measurable using the ping command. Simply use the ping command to "ping" the remote destination and it will report the RTT in milliseconds.

The BDP formula is shown below.

(*B* / 8) * (*T* / 1000)

where, *B* is the bandwidth measured in bits per second and *T* is the RTT measured in milliseconds.

As an example, if the maximum bandwidth is 1 Gbit/s and the RTT is 60 ms, the BDP is calculated as

(1,000,000,000 / 8) * (60 / 1000) = 7,500,000 bytes = 7.5 MB

TCP High Performance Extensions

Originally, TCP/IP was not optimized for transmission links with a high bandwidth delay product (BDP). RFC 1323 *TCP Extensions for High Performance* introduced changes to the TCP protocol to improve performance over high BDP links. RFC 1323 includes a number of TCP changes, but the most relevant one for this discussion is the window scaling option.

The TCP receive window size is negotiated by the TCP implementations during the three-way handshake when the connection is opened. The window specifies the amount of buffer space the receiving TCP has available for data. The TCP sender does not send any more data then the receiver's available window. The TCP window is a form of flow control to prevent the sender from sending more data then the receiver has buffer space available.

The TCP receive window is defined in the TCP header as a 16 bit field, so the maximum window size is 65 KiB. For a transmission link with a large BDP, this is only a fraction of the amount of data the transmission link will hold. Consequentially, the transmission link will never fill to capacity and maximum bandwidth never achieved. RFC 1323 added the window scaling option so that a larger TCP window can be negotiated. The window scaling option makes the TCP window size effectively a 32-bit value, however RFC 1323 does limit it to 1 GiB. The TCP implementations on both sides of the socket connection must support window scaling for it to be used.

TCP Buffers

The TCP receive window used by the TCP on the receiving end of a connection is typically determined from the size of the TCP receive buffer used for the connection. The default TCP receive buffer can typically be configured as part of the TCP configuration. However the default is not typically large enough for high BDP transmission links. The TCP socket API provides an interface for the application to request specific TCP receive and send buffer sizes. The application can request any buffer size and TCP will determine what size it actually uses based on its configuration limits. If TCP on both ends of the socket connection support RFC 1312 window scaling, the TCP window may be as large as 1 GiB if the TCP configuration permits.

TCP Buffer Configuration

In general, the optimum TCP buffer size matches the BDP for the transmission link. However, TCP buffers are maintained by TCP in virtual storage. Very large buffer sizes may actually reduce transmission rates if the virtual storage requirements exceed the system memory capabilities.

Some product components provide configuration options to specify the TCP send and receive buffer sizes. Configuration options TCP_RECV_BUFFER and TCP_SEND_BUFFER specify the TCP receive and send buffer sizes, respectively. The product components on both ends of the TCP socket connection must be configured using the TCP_RECV_BUFFER and TCP_SEND_BUFFER options. Product components typically consist of a Manager component (such as UDM Manager) and a Server component (such as UDM Server). The actual connection between the Manager and Server is established first with the Universal Broker component. A Manager component first establishes a socket connection with the Broker which then starts the Server component and passes the socket connection to the Server. Consequentially, the Universal Broker will always require TCP buffer configuration changes in order to tune network performance of product components.

Network Data Transmission Configurable Options

- Configurable Options
  - CODE_PAGE
  - CTL_SSL_CIPHER_LIST
  - DATA_AUTHENTICATION
  - DATA_COMPRESSION
  - DATA_ENCRYPTION
  - DATA_SSL_CIPHER_LIST
  - DEFAULT_CIPHER
  - ENCRYPT_CONTROL_SESSION
  - KEEPALIVE_INTERVAL
  - NETWORK_DELAY
  - SIO_MODE

Configurable Options

The network protocol can be configured in ways that affect compression, encryption, code pages, and network delays.

The following configuration options are available on many Universal Agent components:

CODE_PAGE

The CODE_PAGE option specifies the code page translation table used to translate network data from and to the local code page for the system on which the program is executing.

A codepage table is text file that contain a two-column table. The table maps local single byte character codes to two-byte UNICODE character codes.

Code pages are located in the product National Language Support (NLS) directory or library. New code pages may be created and added to the NLS directory or library. The CODE_PAGE option value is simply the name of the code page file without any file name extension if present.

CTL_SSL_CIPHER_LIST

The CTL_SSL_CIPHER_LIST option specifies one or more SSL/TLS cipher suites that are acceptable to use for network communications on the control session, which is used for component internal communication.

The SSL/TLS protocol uses cipher suites to specify the combination of encryption and message digest algorithms used for a session. An ordered list of acceptable cipher suites can be specified in a most-to-least order of preference.

An example cipher suite list is RC4-MD5,RC4-SHA,AES128-SHA. The RC4-MD5 cipher suite is the most preferred and AES128-SHA is the least preferred.

When two Universal Agent components (Manager and a Server) first connect, they perform an SSL/TLS handshake that negotiates the cipher suite to use for the session. The Manager presents a list of cipher suites (in descending order of preference) that it would like to use. This is compared against a list of ciphers that the Server supports. The first cipher suite in common is the one used for the session.

DATA_AUTHENTICATION

The DATA_AUTHENTICATION option specifies whether or not the network data is authenticated. Data authentication verifies that the data did not change from the point it was sent to the point it was received.

Data authentication also is referred to as a data integrity in this document.

Data authentication occurs for each message sent over the network. If a message fails authentication, the network session is terminated and both programs end with an error.

The DATA_AUTHENTICATION option is applicable to the **UNVv2** protocol only. SSL/TLS always performs authentication.

DATA_COMPRESSION

The DATA_COMPRESSION option specifies that network data be compressed.

Compression attempts to reduce the amount of data to a form that can be decompressed to its original form. The compression ratio is the original size divided by the compressed size. The compression ratio value will depend on the type of data. Some data compress better than others.

Two methods of compression are available:

1. ZLIB method provides the highest compression ratio with the highest use of CPU
2. HASP method provides the lowest compression ratio with the lowest use of CPU.

Whether or not compression is used and which compression method is used depends on several items:

- Network bandwidth. If network bandwidth is small, compression may be worth the cost in CPU.
- CPU resources. If CPU is limited, the CPU cost may not be worth the reduced bandwidth usage.
- Data compression ratio. If the data does not compress well, it is probably not worth CPU cost. If the data ratio is high, the CPU cost may be worth it.

DATA_ENCRYPTION

The DATA_ENCRYPTION option specifies whether or not network data is encrypted.

Encryption translates data into a format that prevents the original data from being determined. Decryption translates encrypted data back into its original form.

The type of encryption performed depends on the network protocol being used, SSL/TLS or **UNVv2**.

Data encryption does increase CPU usage. Whether or not encryption is used depends on the sensitivity of the data and the security of the two host systems and the data network between the hosts.

DATA_SSL_CIPHER_LIST

The DATA_SSL_CIPHER LIST option specifies one or more SSL/TLS cipher suites that are acceptable to use for network communications on the data session, which is used for standard I/O file transmission.

(See CTL_SSL_CIPHER_LIST.)

DEFAULT_CIPHER

The DEFAULT_CIPHER option specifies the SSL cipher suite to use (since SSL protocol requires a cipher suite) if the DATA_ENCRYPTION option is set to **no**. The default DEFAULT_CIPHER is NULL-MD5 (no encryption, MD5 message digest).

All SSL/TLS cipher suites have a message digest for good reasons. The message digest ensures that the data sent are the data received. Without a message digest, it is possible for bits of the data packet to get changed without being noticed.

ENCRYPT_CONTROL_SESSION

The ENCRYPT_CONTROL_SESSION option is a server-only option that enforces encryption on the control session. When the option is set to a value of no, the server will accept a control session protocol without encryption and message authentication codes (MACs). The default is yes.

Starting with Universal Agent, a manager can request that the UNVv2 protocol be used without encryption or MACs. Considering that host systems may require differing security policies, this option allows for each server to be configured appropriately based on its security policy.

KEEPALIVE_INTERVAL

The KEEPALIVE_INTERVAL option specifies how often, in seconds, a keepalive message (also commonly known as a heartbeat message) is sent between a manager and server.

A keepalive message ensures that the network and both programs are operating normally. Without a keepalive message, error conditions can arise that place one or both programs in an infinite wait.

A keepalive message is sent from the server to the manager. If the server does not receive a keepalive acknowledgement from the manager in a certain period of time (calculated as the maximum of 2 x NETWORK_DELAY or the KEEPALIVE_INTERVAL), the server considers the manager or network as unusable.

How the server processes a keepalive time-out depends on what fault tolerant features are being used. If no fault tolerant features are being used, the server ends with an error. The manager expects to receive a keepalive message in a certain period of time (calculated as the KEEPALIVE_INTERVAL + 2 x NETWORK_DELAY.

NETWORK_DELAY

The NETWORK_DELAY option provides the ability to fine tune Universal Agent network protocol. When a data packet is sent over a TCP/IP network, the time it takes to reach the other end depends on many factors, such as, network congestion, network bandwidth, and the network media type. If the packet is lost before reaching the other end, the other end may wait indefinitely for the expected data.

In order to prevent this situation, Universal Agent components time out waiting for a packet to arrive in a specified period of time. The delay option specifies this period of time.

NETWORK_DELAY specifies the maximum acceptable delay in transmitting data between two programs. Should a data transmission take longer than the specified delay, the operation ends with a time out error. Universal Agent components will consider a time out error as a network fault.

The default NETWORK_DELAY value is 120 seconds. This value is reasonable for most networks and operational characteristics. If the value is too small, false network time outs could occur. If the value is too large, programs will wait a long period of time before reporting a time out problem.

SIO_MODE

The SIO_MODE option specifies whether the data transmitted over the network is processed as text data or binary data.

Text data is translated between the remote and local code pages. Additionally, end of line representations are converted

Text translation operates in two modes: direct and UCS. The default is direct. The direct translation mode exchanges code pages between Universal Agent components to build direct translation tables. Direct translation is the fastest translation method when a significant amount (greater then 10K) of text data is transmitted. The code page exchange increases the amount of data sent over the network as part of the network connection negotiation.

UCS translation does not require the exchange of code pages. For transactions that have little text data transmission, this is the fastest.

Binary data is transmitted without any data translation.

Event Log Dump for Windows

Overview

Universal Agent provides the ability to select records from a Windows event log and write them to a specified output file via its Universal Event Log Dump utility.

All records from a log can be dumped, or event records can be selected according to the date and time that they were generated.

Universal Event Log Dump can be run any time as a stand-alone application. It also is designed to work with Universal Command, which provides centralized control from any operating system and additional options for redirecting output.

Universal Event Log Dump consists of the command line program (**ueld**) followed by a list of configuration options.

Examples

- Execute Universal Event Log Dump from z/OS Manager
- Execute Universal Event Log Dump from a Windows Server

Examples

- Execute Universal Event Log Dump from z/OS Manager
- Execute Universal Event Log Dump from a Windows Server

Execute Universal Event Log Dump from zOS Manager

- Execute Universal Event Log Dump from z/OS Manager
  - Script Options
  - SYSIN Options
  - Components

Execute Universal Event Log Dump from z/OS Manager

The following figure illustrates the execution of Universal Event Log Dump from a z/OS Universal Command Manager.

The application log, from the previous day at 15:00 until current time, will be dumped to the stdout of the manager process to be archived.

```
//S1 EXEC UCMDPRC
//LOGONDD  DD DISP=SHR,DSN=hlq.userid(userid)
//SCRIPTDD DD *
ueld -logtype APPLICATION -stime "*-1,15:00 PM"
//SYSIN    DD *
 -script SCRIPTDD
 -encryptedfile LOGONDD
 -host dallas
/*
```

The JCL procedure **UCMDPRC** is used to execute the **ueld** command. The command is sent to a remote system named **dallas** for execution. The **UNVOUT** DD in the **UCMDPRC** points to sysout, and is where the stdout of the remote command will be written. Additional command line options are read from the encrypted file allocated to DD **LOGONDD**.

Script Options

The script options used in this example are:

Option	Description
-logtype	Event log to be dumped.
-stime	Starting date and time.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	ddname from which to read a script file. The script file is sent to the remote system by UCMD Manager for execution
-encryptedfile	File from which to read an encrypted command options file.
-host	Host name or IP address of the remote system on which to execute the script.

Components

Universal Command Manager for z/OS

Universal Event Log Dump

Execute Universal Event Log Dump from a Windows Server

Execute Universal Event Log Dump from a Windows Server

The following example illustrates the execution of Universal Event Log Dump from a Windows server.

The application log, from the previous day at 15:00 until current time, will be dumped to a file on the server.

```
ueld -logtype APPLICATION -stime "*-1,15:00 PM" -file c:\application.log
```

Command Line Options

The command line options used in this example are:

Command Options	Description
-logtype	Event log to be dumped.
-stime	Starting date and time.
-file	Complete path to the file that will be used to store the selected event log records.

Components

Universal Event Log Dump

z/OS CANCEL Command Support

Overview

Universal Agent provides network fault tolerance and, in some cases, manager fault tolerance (see Fault Tolerance Implementation). These features provide users with the ability to execute jobs that will continue to run when the network is down and when a manager is terminated.

However, there are scenarios in which the user may want to cancel an executing job that supports manager and/or network fault tolerance and have processes terminate immediately. Because of fault tolerance, when the manager is terminated, the server side would begin a connection reestablishment protocol and continue to execute. This would allow the started user job to continue running.

In particular, z/OS supports a CANCEL command that will terminate a job executing on the z/OS operating system.

Detailed Information

The following pages provide detailed information for z/OS CANCEL command support:

- z/OS CANCEL Command Support - Universal Command
- z/OS CANCEL Command Support - Universal Connector
- z/OS CANCEL Command Support - Universal Data Mover

z/OS CANCEL Command Support - Universal Command

- Overview
- Exit Codes
- Security Token

Overview

A user may want to cancel an executing Universal Command job that supports manager and/or network fault tolerance and have both the manager and server processes terminate immediately. Because of fault tolerance, when the manager is terminated, the server side would begin a connection reestablishment protocol and continue to execute. This would allow the started user job to continue running.

When a Universal Command job is cancelled via the z/OS CANCEL command, the job terminates with either of these exit codes:

- Exit code S122, if it is cancelled with a dump.
- Exit code S222, if it is cancelled without a dump.

Part of the responsibility of a Universal Broker executing on a particular host is to monitor the status of all locally running manager processes on that machine. So, when instructed, that Universal Broker could issue a STOP command to the Universal Command Server process associated with the stopped/ended manager process.

Exit Codes

Through the use of the SERVER_STOP_CONDITIONS configuration option, the Universal Command Manager process notifies the locally running Universal Broker of the exit codes that should cause it to terminate the running Server process. With this option, the user can specify a list of exit codes that should trigger the locally running Universal Broker to issue the STOP command to the manager's Universal Command server-side process.

SERVER_STOP_CONDITIONS can specify a single exit code or a comma-separated list of exit codes. These stop conditions are passed from the manager to the locally running Universal Broker, which store this and other component-specific data about the executing manager component. When this executing Universal Command Manager process is cancelled or stopped, the locally running Universal Broker detects the ending of the manager process and retrieves its process completion information, which includes the exit code of the manager.

The Universal Broker then compares this exit code with the list of exit codes provided by SERVER_STOP_CONDITIONS. If a match is found, and either network fault tolerance or manager fault tolerance is enabled, the Universal Broker will execute a **uctl** command to STOP the running Universal Command Server component.

Security Token

For security purposes, Universal Agent passes around a security token that is used by the locally running Universal Broker to STOP associated Universal Command Server process.

This security token is generated on a component-by-component basis by the Universal Broker process that starts the Universal Command Server. Upon generation, this token is returned to the Universal Command Manager which, in turn, updates its locally running Universal Broker with this token. The locally running Universal Broker then uses this token with the issued STOP command to cancel the running Universal Command Server process.

When this token is received by the Universal Broker processes with the request to STOP the server component, the Broker authenticates the received token with the stored token for the running Universal Command Server process. When the token is authenticated, the Universal Command Server process is STOPPED.

z/OS CANCEL Command Support - Universal Connector

Overview

A user may want to cancel an executing Universal Connector job that supports client and / or network fault tolerance and have both the Universal Connector and SAP processes terminate immediately. Because of the separation of work between Universal Connector and SAP, when the Universal Connector client is terminated, the SAP job continues to execute.

In particular, z/OS supports a CANCEL command that will terminate a job executing on the z/OS operating system. When a Universal Connector job is cancelled via the z/OS CANCEL command, the job terminates with either of these exit codes:

- S122, if job is cancelled with a dump.
- S222, if job is cancelled without a dump.

Part of the responsibility of a Universal Broker executing on a particular host is to monitor the status of all locally running manager processes on that machine. So, when instructed, that Universal Broker could invoke a new instance of Universal Connector and issue a CANCEL command to terminate the associated SAP job.

Exit Codes

Through the use of the SERVER_STOP_CONDITIONS configuration option, the Universal Connector process notifies the locally running Universal Broker of the exit codes that should cause it to terminate the running SAP job. With this option, you can specify a list of exit codes that should trigger the locally running Universal Broker to invoke a Universal Connector process to terminate the SAP job.

SERVER_STOP_CONDITIONS can specify a single exit code or a comma-separated list of exit codes. These stop conditions are passed from the manager to the locally running Universal Broker, which stores this and other component-specific data about the executing manager component. When this executing Universal Connector process is cancelled or stopped, the locally running Universal Broker detects the ending of the manager process and retrieves its process completion information, which includes the exit code of the manager.

The Universal Broker then compares this exit code with the list of exit codes provided by SERVER_STOP_CONDITIONS. If a match is found, the Universal Broker will invoke a new instance of the Universal Connector to execute a CANCEL command to terminate the running SAP job.

z/OS CANCEL Command Support - Universal Data Mover

- Overview
- Exit Codes
- Security Token

Overview

When a Universal Data Mover job is cancelled via the z/OS CANCEL command, the job terminates with either of these exit codes:

- Exit code S122, if it is cancelled with a dump.
- Exit code S222, if it is cancelled without a dump.

Part of the responsibility of a Universal Broker executing on a particular host is to monitor the status of all locally running manager processes on that machine. So, when instructed, that Universal Broker could issue a STOP command to the Universal Data Mover Server process associated with the stopped / ended manager process.

In the case of a Universal Data Mover three-party transfer, both the primary and secondary servers need to be cancelled. The Universal Broker running locally with the cancelled Universal Data Mover Manager process will send a STOP command to the primary server. This primary server will, in turn, forward the STOP command to the secondary server, thus cancelling both servers of the three-party transfer.

Exit Codes

Through the use of the SERVER_STOP_CONDITIONS configuration option, the Universal Data Mover Manager process notifies the locally running Universal Broker of the exit codes that should cause it to terminate the running Server process. With this option, the user can specify a list of exit codes that should trigger the locally running Universal Broker to issue the STOP command to the manager's Universal Data Mover server-side process.

SERVER_STOP_CONDITIONS can specify a single exit code or a comma-separated list of exit codes. These stop conditions are passed from the manager to the locally running Universal Broker, which store this and other component-specific data about the executing manager component. When this executing Universal Data Mover Manager process is cancelled or stopped, the locally running Universal Broker detects the ending of the manager process and retrieves its process completion information, which includes the exit code of the manager.

The Universal Broker then compares this exit code with the list of exit codes provided by SERVER_STOP_CONDITIONS. If a match is found, and network fault tolerance is enabled, the Universal Broker will execute a uctl command to STOP the running Universal Data Mover Server component.

Security Token

For security purposes, Universal Agent pass around a security token that is used by the locally running Universal Broker to STOP associated Universal Data Mover Server process.

This security token is generated on a component-by-component basis by the Universal Broker process that starts the Universal Data Mover Server. Upon generation, this token is returned to the Universal Data Mover Manager which, in turn, updates its locally running Universal Broker with this token. The locally running Universal Broker then uses this token with the issued STOP command to cancel the running Universal Data Mover Server process.

When this token is received by the Universal Broker processes with the request to STOP the server component, the Broker authenticates the received token with the stored token for the running Universal Data Mover Server process. When the token is authenticated, the Universal Data Mover Server process is STOPPED.

UAGRERUN Stand-Alone Execution

- Introduction
- Stand-Alone Capabilities and Limitations
- Running UAGRERUN Stand-Alone
  - Explanation of JCL statements:
- UAGRERUN Parameter Descriptions

Introduction

This document describes the procedure for running the UAG for z/OS UAGRERUN program in stand-alone mode.

UAGRERUN is a program included with UAG for z/OS that runs as the first step in any job submitted by UAG for z/OS. Its responsibilities include:

- Creating the control blocks needed for the agent to track the job to completion.
- On rerun, comparing the current JCL to the previous runs' JCL.
- Deleting qualifying datasets created by the job with DISP=NEW
- Performing any GDG dataset adjustments required.
- Performing step selection.
- Printing reports.

Stand-Alone Capabilities and Limitations

UAGRERUN can also be run stand-alone. In that case, it provides similar functionality without any involvement of UAG for z/OS or the Universal Controller.

When running stand-alone, UAGRERUN can provide the following functionality:

- Deleting qualifying datasets created by the job with DISP=NEW (Except GDGs).
- On rerun, performing GDG dataset adjustments if GDGPROC=R is specified.
- Performing step selection.
- Printing reports.

When provided with a history dataset, it can perform the following additional processing:

- On rerun, comparing the current JCL to the previous runs' JCL.
- On rerun, deleting qualifying GDG datasets created by the job with DISP=NEW if GDGPROC=A is specified.

When running stand-alone, the following functionality is unavailable:

- Capturing and using of any run-time information on the job. Including step condition codes, ABENDs, JCL errors and other failures.
- Inheriting parameter setting from UAG for z/OS. This includes, but is not limited to, setting such as dsn_delete_exclude and gdg_processing.
- Any functionality that includes communication to- or from the Universal Controller.
- Rerunning jobs that were originally run through the Universal Controller is not supported.

If a serious error occurs that might compromise the execution of the job, UAGRERUN will issue a message describing the error and issue a U4095 ABEND.

Running UAGRERUN Stand-Alone

In order to run UAGRERUN in stand-alone mode, the Job step below should be added as the first step in the job.

```
//STEP00   EXEC PGM=UAGRERUN,PARMDD=SYSIN
//UNVHIST  DD DSN=<DSNAME>,DISP=(MOD,CATLG),
//         UNIT=SYSALLDA,SPACE=(CYL,(2,2)),
//         DCB=(DSORG=PS,RECFM=VB,BLKSIZE=4104,LRECL=4100)
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
NOUAG,
<Other-parameters>
/*
```

Explanation of JCL statements:

JCL Statement	Description
STEP00	This line executes program UAGRERUN and indicates the parameters can be found in the dataset defined by DDNAME SYSIN. Program UAGRERUN can likely be found in the system link-list, so a STEPLIB or JOBLIB should not be necessary. If needed, a STEPLIB can be added after the EXEC statement.
UNVHIST	This is an optional DD statement to define the history dataset to UAGRERUN. The history dataset is used to store information about this execution of the job. During a rerun, information about the previous run will be read from this dataset. If a rerun of the job might be required, this DD statement is required and must point to the same dataset during rerun as was used during the previous run of the job. The same dataset can be used for multiple jobs as long as they do not require concurrent access. Data will never be deleted from this dataset. A new job record will be added for each job. Occasionally, it might be necessary to delete this dataset as it gets too large.

	The dataset must be a Physical Sequential dataset (DSORG=PS). It can be a fixed or variable record length. Variable is recommended. The record length must be a minimum of 4096 bytes for fixed and 4100 bytes for variable length records. Any valid block size can be used. Space requirements for the dataset are dependent on the number of job steps and the number of datasets used by the job. It is however unlikely that even the largest job would need more than 2 tracks of 3390 type DASD.
SYSPRINT	This DD statement defines the output dataset used by UAGRERUN to write its reports.
SYSIN	Defines the input parameter dataset referenced by the PARMDD keyword on the EXEC statement. A PARM keyword can also be used on the EXEC statement. However, the system limits the total length of all parameters which can be coded on a PARM keyword to 100 bytes. By using PARMDD, this limit is extended to 32,760 bytes.
NOUAG	This parameter is required to run UAGRERUN in stand-alone mode

UAGRERUN Parameter Descriptions

This section describes the parameters used when running UAGRERUN stand-alone.

⚠ Any parameter can be commented out by adding a hashtag ('#') in front of it.

Parameter	Description
NOUAG	This parameter indicates UAGRERUN is to run stand-alone. It is a required parameter.
RERUN=<Job-ID>	Indicates this is a rerun of the job with the specified job ID. The job name cannot have changed between the initial run and the rerun. Information of the job with the specified ID must have been saved in the dataset referenced by the UNVHIST DD statement.
FORCE	Valid with RERUN only. Force rerun ignores JCL changes between the initial run and the rerun.
GDGPROC={ A | R }	Indicates the GDG dataset processing mode. When GDGPROC=A is specified, GDG datasets will be processed in absolute mode. The absolute generation (.G0000V00 format) of GDG datasets will be used during rerun processing. This requires use of the UNVHIST dataset. e.g.: Step STEP01 of a job creates a +1 generation of a GDG dataset. The actual generation produced is MY.SAMPLE.GDG.G0103V00. When the job is rerun, UAGRERUN will delete dataset MY.SAMPLE.GDG.G0103V00 and override the GDG name for STEP01 to recreate MY.SAMPLE.GDG.G0103V00. When GDGPROC=R is specified, GDG datasets will be processed in relative mode. The relative generation number (0, +1 -1, etc.) will be used to process GDG datasets. • Example 1: Step STEP01 of a job creates a +1 generation of a GDG dataset. When the job is rerun, UAGRERUN will not delete the dataset created in the previous run. A new +1 generation will be produced. • Example 2: Step STEP01 of a job creates a +1 generation of a GDG dataset. Step STEP02 reads in the +1 generation and prints reports based on the data. When the job is rerun, step STEP01 is not executed. UAGRERUN will adjust the generation of the GDG in STEP02 to reference generation 0 instead.
OPSDSDEL={ Y | N)	This parameter is used to prevent UAGRERUN from deleting datasets. Default behavior for UAGRERUN is to delete qualifying datasets before executing the job. Setting OPSDSDEL to N prevents UAGRERUN from deleting any datasets.
PREVIEW	This parameter causes UAGRERUN to print its reports only. No datasets will be deleted and no steps other than UAGRERUN will executed.
HELP	UAGRERUN will print instructions and end.
START={ STEP [.PROCSTEP] | (2 – 255) }	This parameter is used to select the first step to execute. The default is the second step. The value of the parameter can be a step name, the name of a step in a procedure, or the absolute number of a step. If the selected step executes a JCL procedure, the first step in the procedure will be the first step executed.
END={ STEP [. PROCSTEP] | (	This parameter is used to select the last step to execute. The default is the last step.

2 – 255) }	The value of the parameter can be a step name, the name of a step in a procedure, or the absolute number of a step. If the selected step executes a JCL procedure, the last step in the procedure will be the last step executed.
SKIP={ STEP [. PROCSTEP] | (2 – 255) }	This parameter is used to skip execution of a step. If the indicated step executes a JCL procedure, none of the steps in the procedure will be executed. SKIP will cause a step to not be executed, even if it has been previously selected to execute using START/END or NOSKIP.
NOSKIP={ STEP [. PROCSTEP] | (2 – 255) }	This parameter is used to force execution of a step. If the indicated step executes a JCL procedure, all steps in the procedure will be executed. NOSKIP will cause a step to be executed, even if it has been previously excluded from execution using START/END or SKIP.

⚠ START, END, SKIP and NOSKIP can be combined to select the job steps to execute. Initially, all steps will be selected for execution. If either START or END is specified, they will be applied first. Then, SKIP and NOSKIP will be applied in the order they are specified. On rerun, any step selections from the prior run will be remembered and honored, unless overridden.

Examples

Execute UAGRERUN in stand-alone mode. Run all steps and delete any datasets before execution.

```
//STEP00   EXEC PGM=UAGRERUN,PARM=NOUAG
//SYSPRINT DD SYSOUT=*
```

Execute UAGRERUN in stand-alone mode. The job has 5 steps, not counting UAGRERUN. Run only the last 3.

```
//STEP00   EXEC PGM=UAGRERUN,PARM='NOUAG,START=4'
//SYSPRINT DD SYSOUT=*
```

Execute UAGRERUN in stand-alone mode. Rerun steps STEP04 through STEP12 of the job with ID JOB29756 but skip step PRTRPT in the procedure executed by step STEP06. Run in preview mode only.

```
//STEP00   EXEC PGM=UAGRERUN,PARMDD=SYSIN
//UNVHIST  DD DSN=MY.SAMPLE.DSNAME,DISP=(MOD,CATLG),
//         UNIT=SYSALLDA,SPACE=(CYL,(2,2)),
//         DCB=(DSORG=PS,RECFM=VB,BLKSIZE=4104,LRECL=4100)
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
NOUAG,
RERUN=JOB29756,
PREVIEW,
START=STEP04,
END=STEP12,
SKIP=STEP06.PRTRPT
/*
```