



# Stonebranch Solutions

Version 4.2.0

Infitran  
User Guide  
infitran-user-4201



---

# Infitran

## User Guide

Stonebranch Solutions 4.2.0

---

Document Name	Infitran 4.2.0 User Guide				
Document ID	infitran-user-4201				
Components	z/OS	UNIX	Windows	IBM i	HP NonStop
Universal Connector	✓	✓			
UNiversal Data Mover	✓	✓	✓	✓	
Universal Enterprise Controller	✓		✓		
Universal Enterprise Controller Client Applications			✓		
Universal Event Monitor	✓	✓	✓		
Universal Event Monitor for SOA		✓	✓		
Universal Certificate	✓	✓	✓		
Universal Control	✓	✓	✓	✓	
Universal Database Dump	✓	✓	✓		
Universal Database Load	✓	✓	✓		
Universal Display Log File				✓	
Universal Encrypt	✓	✓	✓	✓	
Universal Event Log Dump			✓		
Universal Message Translator	✓	✓	✓	✓	
Universal Query	✓	✓	✓	✓	
Universal Return Code			✓		
Universal Spool List	✓	✓	✓	✓	
Universal Spool Remove	✓	✓	✓	✓	
Universal Submit Job				✓	
Universal Write to Operator	✓				

---

# Stonebranch Documentation Policy

---

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

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

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

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

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

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

© 2008-2010 by Stonebranch, Inc.

All rights reserved.



---

# Summary of Changes

---

Changes for Infitran 4.2.0 User Guide  
(infitran-user-4201)  
October 29, 2010

- Created links from Stonebranch Solutions components to their corresponding reference guides in Section [1.4 Infitran Components](#).
- Removed requirement for licensed version of Universal Command in Section [3.1 Overview](#) of Chapter [3 Remote Execution](#).
- Replaced z/OS examples with Windows and UNIX examples in Chapter [14 Databases](#).

Changes for Infitran 4.2.0 User Guide  
(infitran-user-4200)  
August 6, 2010

- This is the first version of the Infitran 4.2.0 User Guide. Information on Infitran features, and examples of how those features can be implemented, have been moved from Universal Products 4.1.0 user guides to this document.

---

# Contents

---

Summary of Changes .....	5
Contents .....	6
List of Figures .....	20
List of Tables .....	25
Preface .....	27
Document Structure .....	27
Cross-Reference Links .....	27
Conventions .....	28
Document Organization .....	30
1 Infitran Overview .....	32
1.1 What is Infitran? .....	32
1.2 What can Infitran do for Me? .....	33
1.3 Infitran Features .....	34
1.4 Infitran Components .....	36
Universal Data Mover .....	36
Universal Event Monitor .....	36
Universal Event Monitor for SOA .....	37
Universal Enterprise Controller .....	37
Universal Event Subsystem .....	37
Universal Enterprise Controller Client Applications .....	38

I-Activity Monitor .....	38
I-Management Console .....	38
I-Administrator .....	38
Universal Broker .....	38
Stonebranch Solutions Utilities .....	39
Universal Certificate .....	39
Universal Control .....	39
Universal Database Dump .....	39
Universal Database Load .....	39
Universal Display Log File .....	39
Universal Encrypt .....	39
Universal Event Log Dump .....	40
Universal Message Translator .....	40
Universal Install Merge .....	40
Universal Query .....	40
Universal Return Code .....	40
Universal Spool List .....	40
Universal Spool Remove .....	40
Universal Submit Job .....	41
Universal Write-to-Operator .....	41
1.5 Limited Use Components .....	42
Universal Command .....	42
Limitations of Use .....	42
Universal Command Agent for SOA .....	42
Limitations of Use .....	42
Universal Connector .....	43
Limitations of Use .....	43
2 Transferring Files to / from Remote Systems .....	44
2.1 Overview .....	44
2.2 Transfer Operation Components .....	45
2.2.1 Manager .....	45
2.2.2 Primary Server .....	45
2.2.3 Secondary Server .....	45
2.3 Transfer Sessions .....	46
2.3.1 Logical Names .....	46
2.3.2 Two-Party Transfer Sessions .....	46
2.3.3 Three-Party Transfer Sessions .....	46
2.4 Transferring Files Examples .....	48

z/OS .....	48
UNIX and Windows .....	48
IBM i .....	49
2.4.1 Copy a File to an Existing z/OS Sequential Data Set .....	50
2.4.2 Copy a z/OS Sequential Data Set to a File .....	52
2.4.3 Copy a Set of Files to an Existing z/OS Partitioned Data Set .....	53
2.4.4 Copy a File to a New z/OS Sequential Data Set .....	55
2.4.5 Copy a Set of Files to a New z/OS Partitioned Data Set .....	56
2.4.6 Simple File Copy to the Manager .....	57
2.4.7 Simple File Copy to the Server .....	58
2.4.8 Copy a Set of Files .....	59
2.4.9 Copy a File to an Existing IBM i File .....	60
2.4.10 Copy an IBM i Data Physical File to a File .....	62
2.4.11 Copy a Set of Files to an Existing Data Physical File .....	63
2.4.12 Copy a File to a New IBM i Data Physical File .....	64
2.4.13 Copy a File to a New IBM i Source Physical File .....	65
2.4.14 Copy a Set of Files to a New Data Physical File on IBM i .....	66
2.4.15 Copy Different Types of IBM i Files using forfiles and \$_file.type .....	67
2.4.16 Invoke a Script from a Batch Job .....	68
3 Remote Execution .....	69
3.1 Overview .....	69
3.2 Execution Primer .....	71
3.2.1 Remote Execution Requirements .....	72
3.3 Remote Execution Examples .....	77
Windows .....	77
UNIX .....	77
IBM i .....	77
3.3.1 Windows Directory Listing Using a Batch File - Default Directory .....	78
3.3.2 Windows Directory Listing Using a Batch File - Returned File .....	80
3.3.3 UNIX Listing Using a Shell Script .....	82
3.3.4 UNIX - Integrating UDM with FTP Using a Shell Script .....	85
3.3.5 UNIX - Integrating UDM with FTP Using a Command Reference .....	87
3.3.6 IBM i from Windows, UNIX, or IBM i - exec Command Return Codes .....	89
4 Remote Execution for SAP Systems .....	92
4.1 Overview .....	92
4.2 Remote Execution of SAP Examples .....	93
z/OS .....	93



UNIX .....	93
4.2.1 Raising an SAP Event for z/OS Example .....	94
4.2.2 Raising an SAP Event for UNIX Example .....	96
5 Web Services Execution .....	98
5.1 Overview .....	98
5.2 Web Services Examples .....	99
UNIX .....	99
5.2.1 Inbound JMS Examples .....	100
ActiveMQ Topic .....	101
Websphere Queue .....	102
MQ Series Queue: .....	103
Triggering an Event .....	104
5.2.2 Inbound SOAP Implementation .....	106
6 Event Monitoring and File Triggering .....	111
6.1 Overview .....	111
6.2 Universal Event Monitor .....	112
6.2.1 Storing Event Definitions and Event Handlers .....	114
6.2.2 Monitoring a Single Event .....	116
6.2.3 Monitoring Multiple Events .....	118
6.3 UEMLoad .....	120
6.3.1 Controlling Database Access .....	121
Access via UEMLoad Utility .....	121
Universal Access Control List .....	122
6.4 Event Monitoring and File Triggering Examples .....	123
Universal Event Monitoring Examples .....	123
z/OS .....	123
Windows .....	123
UNIX .....	124
6.4.1 Starting an Event-Driven Server .....	125
6.4.2 Refreshing an Event-Driven UEM Server .....	126
6.4.3 Using a Stored Event Handler Record in z/OS .....	127
6.4.4 Handling an Event With a Script in z/OS .....	128
6.4.5 Handling an Expired Event in z/OS .....	130
6.4.6 Continuation Character - in z/OS Handler Script .....	131
6.4.7 Continuation Character + in z/OS Handler Script .....	132
6.4.8 Continuation Characters - and + in z/OS Handler Script .....	133
6.4.9 Using a Stored Event Handler Record in Windows .....	134

6.4.10	Executing a Script for a Triggered Event Occurrence in Windows	135
6.4.11	Handling an Expired Event in Windows	137
6.4.12	Adding a Single Event Record for Windows	138
6.4.13	Adding a Single Event Handler Record for Windows	139
6.4.14	Listing All Event Definitions for Windows	140
6.4.15	Exporting the Event Definition and Event Handler Databases for Windows	141
6.4.16	List a Single Event Handler Record for Windows	142
6.4.17	Listing Multiple Event Definitions and Event Handlers Using Wildcards for Windows	143
6.4.18	Add Record(s) Using a Definition File for Windows	144
6.4.19	Add Record(s) Remotely, Using a Definition File Redirected from STDIN for Windows	145
6.4.20	Add Record(s) Remotely, Using a Definition File Redirected from STDIN (for z/OS) for Windows	146
6.4.21	Definition File Format for Windows	147
6.4.22	Using a Stored Event Handler Record in UNIX	149
6.4.23	Executing a Script for a Triggered Event Occurrence in UNIX	150
6.4.24	Handling an Expired Event in UNIX	152
6.4.25	Adding a Single Event Record for UNIX	153
6.4.26	Adding a Single Event Handler Record for UNIX	154
6.4.27	Listing All Event Definitions for UNIX	155
6.4.28	Exporting the Event Definition and Event Handler Databases for UNIX	156
6.4.29	List a Single Event Handler Record for UNIX	157
6.4.30	Listing Multiple Event Definitions and Event Handlers Using Wildcards for UNIX	158
6.4.31	Add Record(s) Using a Definition File for UNIX	159
6.4.32	Add Record(s) Remotely, Using a Definition File Redirected from STDIN for UNIX	160
6.4.33	Add Record(s) Remotely, Using a Definition File Redirected from STDIN (for z/OS) for UNIX	161
6.4.34	Definition File Format for UNIX	162
7	Security	164
7.1	Overview	164
7.2	Security of Infitran Components	165
7.2.1	Universal Broker	166
	File Permissions	166
	Configuration Files	167
	Universal Access Control List	167

Universal Broker User Account .....	168
7.2.2 Universal Data Mover Manager Security .....	170
File Permissions .....	170
Configuration Files .....	170
7.2.3 Universal Data Mover Server .....	171
File Permissions .....	171
Configuration Files .....	171
Universal Data Mover Server User ID .....	172
Universal Data Mover Server User Profile .....	172
User Authentication .....	172
7.2.4 Universal Event Monitor Manager .....	174
File Permissions .....	174
Data Privacy .....	174
RACF Protection .....	174
Configuration Files .....	175
7.2.5 Universal Event Monitor Server .....	176
Data Privacy .....	176
File Permissions .....	176
Configuration Files .....	176
User Authentication .....	177
7.2.6 Universal Control Manager .....	178
File Permissions .....	178
Configuration Files .....	178
Universal Configuration Manager .....	178
RACF Protection .....	179
7.2.7 Universal Control Server .....	180
File Permissions .....	180
Configuration Files .....	180
Universal Control Server User ID .....	180
User Authentication .....	181
7.2.8 Universal Event Log Dump .....	182
Event Log Access .....	182
Configuration Files .....	182
7.2.9 Universal Spool List .....	182
7.2.10 Universal Spool Remove .....	182
7.3 Encryption .....	183
7.3.1 Encrypting Password Files .....	183
7.3.2 Transferring Encrypted Password Files between Servers .....	184
Security Considerations .....	184
7.4 Encryption Examples .....	185

z/OS .....	185
Windows .....	185
UNIX .....	185
IBM i .....	185
7.4.1 Creating Encrypted Files for z/OS .....	186
7.4.2 Using Encrypted Password File on z/OS .....	187
7.4.3 Creating Encrypted Files for Windows .....	188
7.4.4 Using Encrypted Password File on Windows .....	189
7.4.5 Creating Encrypted Files for UNIX .....	190
7.4.6 Using Encrypted Password File on UNIX .....	191
7.4.7 Creating Encrypted Files for IBM i .....	192
7.4.8 Using Encrypted Password File on IBM i .....	193
7.5 Universal Access Control List .....	194
7.5.1 UACL Configuration .....	195
7.5.2 UACL Entries .....	196
Client Identification .....	196
Request Identification .....	200
Certificate-Based and Non Certificate-Based UACL Entries .....	201
7.6 Universal Access Control List Examples .....	202
z/OS .....	202
Windows .....	202
UNIX .....	202
IBM i .....	202
7.6.1 Universal Broker for z/OS .....	203
7.6.2 Universal Data Mover Server for z/OS .....	204
7.6.3 Universal Control Server for z/OS .....	205
7.6.4 Universal Broker for Windows .....	206
7.6.5 Universal Data Mover Server for Windows .....	207
7.6.6 Universal Control Server for Windows .....	208
7.6.7 Universal Broker for UNIX .....	209
7.6.8 Universal Data Mover Server for UNIX .....	210
7.6.9 Universal Control Server for UNIX .....	211
7.6.10 Universal Broker for IBM i .....	212
7.6.11 Universal Data Mover Server for IBM i .....	213
7.6.12 Universal Control Server for IBM i .....	214
7.7 X.509 Certificates .....	215
7.7.1 Sample Certificate Directory .....	216
7.7.2 Sample X.509 Certificate .....	217
7.7.3 Certificate Fields .....	218
7.7.4 SSL Peer Authentication .....	219

Certificate Verification .....	219
Certificate Revocation .....	219
Certificate Identification .....	220
Certificate Support .....	220
7.8 Creating Certificates Examples .....	221
7.8.1 Creating a Certificate Authority Certificate .....	222
7.8.2 Creating a Certificate .....	223
8 Configuration Management .....	224
8.1 Overview .....	224
8.2 Configuration Methods .....	225
Universal Broker / Servers Configuration Method .....	225
8.2.1 Command Line .....	226
8.2.2 Command Line File .....	228
8.2.3 Environment Variables .....	229
8.2.4 Configuration File .....	231
Configuration File Syntax .....	233
8.3 Remote Configuration .....	234
8.3.1 Unmanaged Mode .....	234
8.3.2 Managed Mode .....	235
Selecting Managed Mode .....	235
8.3.3 Universal Broker Start-up .....	237
8.4 Universal Configuration Manager .....	238
8.4.1 Availability .....	238
8.4.2 Accessing the Universal Configuration Manager .....	240
8.4.3 Navigating through Universal Configuration Manager .....	242
8.4.4 Modifying / Entering Data .....	242
Rules for Modifying / Entering Data .....	242
8.4.5 Saving Data .....	243
8.4.6 Accessing Help Information .....	243
8.4.7 Universal Data Mover Installed Components .....	244
Universal Data Mover Manager .....	244
Universal Data Mover Server .....	245
8.4.8 Universal Event Monitor Installed Components .....	246
Universal Event Monitor Manager .....	246
Universal Event Monitor Server .....	247
8.4.9 Universal Enterprise Controller Component .....	248
8.4.10 Universal Broker Installed Component .....	249
8.5 Configuration Refresh .....	250

8.5.1	Configuration Refresh Via Universal Control	251
	Configuration Refresh for Universal Event Monitor Server	251
8.5.2	Configuration Refresh Via Universal Configuration Manager	252
8.5.3	Universal Broker Configuration Options Refresh	252
8.6	Refreshing via Universal Control Examples	253
z/OS		253
Windows		253
UNIX		253
IBM i		253
8.6.1	Refreshing Universal Broker from z/OS	254
8.6.2	Refreshing a Component from z/OS	256
8.6.3	Refreshing Universal Broker via Universal Control from Windows	257
8.6.4	Refreshing a Component via Universal Control from Windows	258
8.6.5	Refreshing Universal Broker via Universal Control from UNIX	259
8.6.6	Refreshing a Component via Universal Control from UNIX	260
8.6.7	Refreshing Universal Broker via Universal Control from IBM i	261
8.6.8	Refreshing a Component via Universal Control from IBM i	262
8.7	Merging Configuration Options during an Upgrade Installation Examples	263
Windows and UNIX		263
Files Used in Examples		264
8.7.1	Merge Files Using Program Defaults	265
8.7.2	Merge Files Introducing New Options	266
8.7.3	Merge Files Using Installation-Dependent Values	267
9	Component Management	268
9.1	Overview	268
9.2	Component Definition	269
9.2.1	Universal Event Monitor Component Definition	269
9.3	Starting Components	270
Starting Manually		270
Start via Manager		270
Starting Automatically		270
Starting via Universal Control		270
9.4	Stopping Components	271
9.5	Starting / Stopping Universal Broker Examples	272
z/OS		272
Windows		272
UNIX		272
IBM i		272

9.5.1 Starting / Stopping Universal Broker for z/OS	273
Start Universal Broker	273
Stop Universal Broker	273
9.5.2 Starting Universal Broker for Windows	274
Console Application	274
Windows Service	274
9.5.3 Starting Universal Broker for UNIX	275
Daemon	275
Console Application	276
9.5.4 Starting, Ending and Working With Universal Broker for IBM i	277
Commands	278
9.6 Starting / Stopping Components via Universal Control Examples	279
z/OS	279
Windows	279
UNIX	279
IBM i	279
9.6.1 Starting a z/OS Component via Universal Control	280
9.6.2 Stopping a z/OS Component via Universal Control	281
9.6.3 Starting a Windows Component via Universal Control	282
9.6.4 Stopping a Windows Component via Universal Control	283
9.6.5 Starting a UNIX Component via Universal Control	284
9.6.6 Stopping a UNIX Component via Universal Control	285
9.6.7 Starting an IBM i Component via Universal Control	286
9.6.8 Stopping an IBM i Component via Universal Control	287
9.7 Maintaining Universal Broker Definitions in the Universal Enterprise	
Controller Database	288
z/OS and Windows	288
z/OS	288
Windows	288
9.7.1 List All Defined Brokers	289
9.7.2 Export a Specific Defined Broker	289
9.7.3 Export Events	289
9.7.4 Retrieve Archived File and Export	290
9.7.5 Delete a Specific Defined Broker	290
9.7.6 Add Specific Defined Broker via deffile	291
9.7.7 Add Existing Brokers to a Broker Group	292
9.7.8 Delete Existing Brokers to a Broker Group	292
9.7.9 Export Events into ARC Format for z/OS	293
9.7.10 Retrieve Archived File and Export into XML for z/OS	293

9.7.11	Export Events into ARC Format for Windows	294
9.7.12	Retrieve Archive File and Export into CSV for Windows	294
10	Messaging and Auditing	295
10.1	Overview	295
10.2	Messaging	296
10.2.1	Message Types	296
10.2.2	Message ID	297
10.2.3	Message Levels	297
10.2.4	Message Destinations	298
z/OS	Message Destinations	298
UNIX	Message Destinations	298
Windows	Message Destinations	299
IBM i	Message Destinations	299
10.3	Auditing	300
10.4	Creating Write-to-Operator Messages Examples	301
z/OS		301
10.4.1	USS UWTO for z/OS Console	302
10.4.2	USS UWTO for z/OS Console and Wait for Reply	303
11	Message Translation	304
11.1	Overview	304
11.2	Usage	305
11.2.1	Translation Table	305
Translation Table Format		305
Translation Table Fields		306
11.2.2	Matching Algorithm	306
11.3	Message Translation Examples	307
All Operating Systems		307
z/OS		307
Windows		307
UNIX		307
IBM i		307
11.3.1	Universal Message Translator (Part 1)	308
11.3.2	Universal Message Translator (Part 2)	309
11.3.3	Execute Universal Message Translator from z/OS	310
11.3.4	Execute Universal Message Translator from Windows	311
11.3.5	Execute Universal Message Translator from UNIX	312
11.3.6	Execute Universal Message Translator from IBM i	313



---

12 Monitoring and Alerting .....	314
12.1 Overview .....	314
12.2 Monitoring of All Agents .....	315
12.2.1 Monitored Information .....	315
12.2.2 Polling .....	315
12.2.3 Alerts .....	315
Alert Types .....	316
12.3 Querying for Job Status and Activity .....	317
12.4 Querying for Job Status and Activity Examples .....	318
All Operating Systems .....	318
z/OS .....	318
UNIX and Windows .....	318
IBM i .....	318
12.4.1 Universal Query Output .....	319
12.4.2 Universal Query for z/OS .....	320
12.4.3 Universal Query for UNIX and Windows .....	321
12.4.4 Universal Query for IBM i .....	322
13 Windows Event Log Dump .....	323
13.1 Overview .....	323
13.2 Windows Event Log Dump Examples .....	324
Windows .....	324
13.2.1 Execute Universal Event Log Dump from z/OS Manager .....	325
13.2.2 Execute Universal Event Log Dump from a Windows Server .....	327
14 Databases .....	328
14.1 Overview .....	328
14.2 Component Information Database .....	329
14.3 Universal Event Monitor Databases .....	330
14.3.1 Event Definition Database .....	331
14.3.2 Event Handler Database .....	332
14.3.3 Event Spool Database .....	333
14.3.4 Controlling UEM Database Access .....	334
14.4 Universal Enterprise Controller Databases .....	335
14.4.1 Database Files .....	335
14.4.2 Database Management .....	335
Automated Database Cleanup .....	335
Memory Management .....	336

---

14.5 Database Backup and Recovery .....	337
14.5.1 Database Backups .....	337
14.5.2 General Database Recovery Procedures .....	338
14.5.3 Database Recovery for Universal Broker .....	339
z/OS .....	339
UNIX .....	339
Windows .....	340
IBM i .....	340
14.5.4 Database Recovery for Universal Enterprise Controller .....	341
z/OS .....	341
Windows .....	342
14.6 Listing Infitran Database Records Examples .....	343
Windows and UNIX .....	343
14.6.1 List Universal Broker Database .....	344
Windows .....	344
UNIX .....	344
14.6.2 List Universal Event Monitor Spool Database Records .....	345
Windows .....	345
UNIX .....	345
14.6.3 List Broker Detail for a Component .....	346
Windows .....	346
UNIX .....	346
14.6.4 List Standard Out for a Component .....	347
Windows .....	347
UNIX .....	347
14.7 Removing Infitran Database Records .....	348
Windows and UNIX .....	348
14.7.1 Remove Component Records .....	349
Windows .....	349
UNIX .....	349
14.7.2 Remove Component Records: Change Broker Database Directory ...	350
Windows .....	350
UNIX .....	350
15 Fault Tolerance Implementation .....	351
15.1 Overview .....	351
15.2 Network Fault Tolerance .....	352
15.2.1 Open Retry .....	353
15.2.2 Component Management .....	353

16 Network Data Transmission .....	355
16.1 Overview .....	355
16.1.1 Secure Socket Layer Protocol .....	356
Data Privacy and Integrity .....	356
Peer Authentication .....	358
16.1.2 Stonebranch Solutions Protocol .....	359
Data Privacy and Integrity .....	359
16.1.3 Stonebranch Solutions Application Protocol .....	360
Low-Overhead .....	360
Secure .....	360
Extensible .....	361
16.1.4 Configurable Options .....	362
17 z/OS Cancel Command Support .....	366
17.1 Overview .....	366
17.1.1 Exit Codes .....	367
17.1.2 Security Token .....	367
A Glossary .....	368
B Customer Support .....	374
Index .....	375

---

# List of Figures

---

2	Transferring Files to / from Remote Systems .....	44
	Figure 2.1    Infitran Transfer Sessions .....	47
3	Remote Execution .....	69
	Figure 3.1    Remote Execution Components: UCMD Manager and Server .....	70
	Figure 3.2    Remote z/OS Execution Using an IP Address .....	73
	Figure 3.3    Remote z/OS Execution Using a Host Name .....	73
	Figure 3.4    Remote z/OS Execution Using a UDM Logical Session Name .....	73
	Figure 3.5    Remote Windows Execution Using an IP Address .....	74
	Figure 3.6    Remote Windows Execution Using a Host Name .....	74
	Figure 3.7    Remote Windows Execution Using a UDM Logical Session Name .....	74
	Figure 3.8    Remote UNIX Execution Using an IP Address .....	75
	Figure 3.9    Remote UNIX Execution Using a Host Name .....	75
	Figure 3.10   Remote UNIX Execution Using a UDM Logical Session Name .....	75
	Figure 3.11   Remote IBM i Execution Using an IP Address .....	76
	Figure 3.12   Remote IBM i Execution Using a Host Name .....	76
	Figure 3.13   Remote IBM i Execution Using a UDM Logical Session Name .....	76
	Figure 3.14   exec Command Under Windows - Listing in Default Directory .....	78
	Figure 3.15   Listing in Default Directory - Example Batch File .....	78
	Figure 3.16   Listing in Default Directory - Listing Sent to stdout.txt .....	78
	Figure 3.17   Listing in Default Directory - Transaction log from UDM via stdout .....	79
	Figure 3.18   exec Command Under Windows - Returned File .....	80
	Figure 3.19   Returned File - Example Batch File .....	81
	Figure 3.20   Returned File - Listing Sent to stdout.txt .....	81
	Figure 3.21   UNIX Listing - UDM Script on Local System .....	82
	Figure 3.22   UNIX Listing - Shell Script on Remote System .....	83
	Figure 3.23   UNIX Listing - Listing Sent to stdout.txt .....	83
	Figure 3.24   UNIX Listing - UDM Manager Transaction Log .....	83
	Figure 3.25   Integrating UDM with FTP - UDM Script on Local System .....	86
	Figure 3.26   Integrating UDM with FTP - Shell Script on Remote System .....	86
	Figure 3.27   Using a Command Reference - UDM Script on Local System .....	87
	Figure 3.28   Using a Command Reference - Command Reference on Remote System .....	88

	Figure 3.29	UDM Script - Exec Command Return Codes .....	90
4	Remote Execution for SAP Systems .....		92
	Figure 4.1	Raising an SAP Event for z/OS - JCL (1 of 2) .....	94
	Figure 4.2	Raising an SAP Event for z/OS - JCL (2 of 2) .....	95
	Figure 4.3	Raising an SAP Event for UNIX - UDM Script File: BIVehicle001 (1 of 2) .....	96
	Figure 4.4	Raising an SAP Event for UNIX - UDM Script File: BIVehicle001 (2 of 2) .....	97
5	Web Services Execution .....		98
	Figure 5.1	Inbound JMS - Constructng Connection to Target .....	100
	Figure 5.2	Inbound JMS - Attachment to an Apache ActiveMQ Dynamic Topic .....	101
	Figure 5.3	Inbound JMS - Attachment to an IBM Websphere Queue .....	102
	Figure 5.4	Inbound JMS - Attachment to an IBM MQ Series Queue .....	103
	Figure 5.5	Triggering an Event .....	104
	Figure 5.6	Inbound SOAP Request UAC.xml .....	106
	Figure 5.7	Inbound SOAP Request – Message Payload Written to <b>process_%Seq%.xml</b> File .....	107
	Figure 5.8	Inbound SOAP Request – Universal Event Monitor Event Definition .....	107
	Figure 5.9	Inbound SOAP Request – Universal Event Monitor Handler Definition .....	108
	Figure 5.10	Outbound SOAP Request – abc.rexx .....	109
	Figure 5.11	Outbound SOAP Request – Event and Handler to purge abc.log .....	110
6	Event Monitoring and File Triggering .....		111
	Figure 6.1	High-Level Interaction of UEM Components .....	113
	Figure 6.2	UEMLoad Utility Overview .....	115
	Figure 6.3	UEM Manager Overview .....	117
	Figure 6.4	UEM Server Overview .....	119
	Figure 6.5	Starting a UEM Event-Driven Server .....	125
	Figure 6.6	Refreshing a UEM Event-Driven Server .....	126
	Figure 6.7	Using a Stored Event Record .....	127
	Figure 6.8	Handling an Event with a Script .....	129
	Figure 6.9	Handling an Expired Event .....	130
	Figure 6.10	Continuation Character "-" in z/OS Handler Script .....	131
	Figure 6.11	Continuation Character "+" in z/OS Handler Script .....	132
	Figure 6.12	Continuation Characters "-" and "+" in z/OS Handler Script .....	133
	Figure 6.13	Using a Stored Event Handler Record .....	134
	Figure 6.14	Handling an Event with a Script. ....	135
	Figure 6.15	Contents of Sample Script File .....	135
	Figure 6.16	Handling an Expired Event .....	137
	Figure 6.17	Adding a single event definition record. ....	138
	Figure 6.18	Adding a single event handler record. ....	139
	Figure 6.19	Listing all event definition records. ....	140
	Figure 6.20	Exporting all Event and Handler Records .....	141
	Figure 6.21	List a Single Event Handler Record .....	142
	Figure 6.22	Sample List Output .....	142
	Figure 6.23	Using Wildcards to List Records .....	143
	Figure 6.24	Add Database Record(s) Using a Definition File .....	144
	Figure 6.25	Redirect Definition File from stdin .....	145

Figure 6.26	Redirect Definition File from STDIN (for z/OS)	146
Figure 6.27	Definition File Sample - Windows	148
Figure 6.28	Using a Stored Event Handler Record	149
Figure 6.29	Handling an Event with a Script	150
Figure 6.30	Contents of Sample Script File	151
Figure 6.31	Handling an Expired Event	152
Figure 6.32	Adding a single event definition record.	153
Figure 6.33	Adding a single event handler record.	154
Figure 6.34	Listing all event definition records.	155
Figure 6.35	Exporting all Event and Handler Records	156
Figure 6.36	List a Single Event Handler Record	157
Figure 6.37	Sample List Output	157
Figure 6.38	Using Wildcards to List Records	158
Figure 6.39	Add Database Record(s) Using a Definition File	159
Figure 6.40	Redirect Definition File from stdin	160
Figure 6.41	Redirect Definition File from STDIN (for z/OS)	161
Figure 6.42	Definition File Sample - UNIX	163
7	Security	164
Figure 7.1	z/OS -encryptedfile example	187
Figure 7.2	Windows -encryptedfile example	189
Figure 7.3	UNIX -encryptedfile example	191
Figure 7.4	IBM i -encryptedfile example	193
Figure 7.5	Universal Configuration Manager - Universal Broker - Access ACL	206
Figure 7.6	Universal Configuration Manager - Universal Data Mover Server - Access ACL	207
Figure 7.7	Universal Configuration Manager - Universal Control Server - Access ACL	208
Figure 7.8	X.500 Directory (sample)	216
Figure 7.9	X.509 Version 3 Certificate (sample)	217
8	Configuration Management	224
Figure 8.1	Remote Configuration - Unmanaged and Managed Modes of Operation	236
Figure 8.2	Universal Configuration Manager Error dialog – Windows Vista / Windows 7	238
Figure 8.3	Program Compatibility Assistant – Windows Vista / Windows 7	239
Figure 8.4	Universal Configuration Manager	241
Figure 8.5	Universal Configuration Manager - UDM Manager	244
Figure 8.6	Universal Configuration Manager - UDM Server	245
Figure 8.7	Universal Configuration Manager - UEM Manager	246
Figure 8.8	Universal Configuration Manager - UEM Server	247
Figure 8.9	Universal Configuration Manager - Universal Enterprise Controller	248
Figure 8.10	Universal Configuration Manager - Universal Broker	249
Figure 8.11	Refreshing Universal Broker via Universal Control from z/OS	254
Figure 8.12	Refreshing Component via Universal Control from z/OS	256
Figure 8.13	Refreshing Universal Broker via Universal Control from Windows	257
Figure 8.14	Refreshing Component via Universal Control from Windows	258
Figure 8.15	Refreshing Universal Broker via Universal Control from UNIX	259
Figure 8.16	Refreshing Component via Universal Control from UNIX	260
Figure 8.17	Refreshing Universal Broker via Universal Control from IBM i	261

Figure 8.18	Refreshing Component via Universal Control from IBM i .....	262
Figure 8.19	Merge infile.txt into outfile.txt using program defaults .....	265
Figure 8.20	Merge infile.txt into outfile.txt keeping new options .....	266
Figure 8.21	Merge infile.txt into outfile.txt using installation-dependent values .....	267
9	Component Management .....	268
Figure 9.1	Universal Broker for UNIX - Daemon Startup Script Syntax .....	275
Figure 9.2	Universal Broker for IBM i - Subsystem Start Command .....	278
Figure 9.3	Universal Broker for IBM i - Subsystem End Command .....	278
Figure 9.4	Universal Broker for IBM i - Subsystem Work With Command .....	278
Figure 9.5	Universal Control for z/OS - Start Component Example .....	280
Figure 9.6	Universal Control for z/OS - Stop Example .....	281
Figure 9.7	Universal Control for Windows - Start Component Example .....	282
Figure 9.8	Universal Control for Windows - Stop Component Example .....	283
Figure 9.9	Start Component Example. ....	284
Figure 9.10	Universal Control Manager for UNIX - Stop Component Example 1 .....	285
Figure 9.11	Start Component Example .....	286
Figure 9.12	Universal Control for IBM i - Stop Component Example .....	287
Figure 9.13	UECLoad - List All Defined Brokers .....	289
Figure 9.14	UECLoad - Export a Specific Defined Broker .....	289
Figure 9.15	UECLoad - Export Events .....	289
Figure 9.16	UECLoad - Retrieve Archived File and Export .....	290
Figure 9.17	UECLoad - Delete a Specific Defined Broker .....	290
Figure 9.18	UECLoad - Add Specific Defined Broker via a Definition File .....	291
Figure 9.19	UECLoad - Definition File used for Adding Specific Defined Broker .....	291
Figure 9.20	UECLoad - Add Existing Brokers to a Broker Group .....	292
Figure 9.21	UECLoad - Delete Existing Brokers to a Broker Group .....	292
Figure 9.22	UECLoad for z/OS - Export Events into ARC Format .....	293
Figure 9.23	UECLoad for z/OS- Retrieve Archived File and Export into XML .....	293
Figure 9.24	UECLoad for Windows - Export Events into ARC Format .....	294
Figure 9.25	UECLoad for Windows - Retrieve Archived File and Export into CSV .....	294
10	Messaging and Auditing .....	295
Figure 10.1	Universal WTO - Issue WTO to z/OS Console .....	302
Figure 10.2	Universal WTO - Issue WTOR to z/OS Console .....	303
11	Message Translation .....	304
Figure 11.1	Universal Message Translator - Example 1, Message File .....	308
Figure 11.2	Universal Message Translator - Example 1, Translation Table 1 .....	308
Figure 11.3	Universal Message Translator - Example 1, Translation Table 2 .....	308
Figure 11.4	Universal Message Translator - Example 2, Message File .....	309
Figure 11.5	Universal Message Translator - Example 2, Translation Table 1 .....	309
Figure 11.6	Universal Message Translator - Execute from z/OS .....	310
Figure 11.7	Universal Message Translator - Execute from Windows .....	311
Figure 11.8	Universal Message Translator - Execute from UNIX .....	312
Figure 11.9	Universal Message Translator - Execute from IBM i .....	313

12	Monitoring and Alerting .....	314
	Figure 12.1 Universal Query Output .....	319
	Figure 12.2 Universal Query for z/OS - Listing Active Components .....	320
	Figure 12.3 Universal Query for UNIX and Windows - Listing Active Components .....	321
	Figure 12.4 Universal Query for IBM i (specific port) - Listing Active Components .....	322
	Figure 12.5 Universal Query for IBM i (default port) - Listing Active Components .....	322
13	Windows Event Log Dump .....	323
	Figure 13.1 Universal Event Log Dump - Execution from z/OS Manager .....	325
	Figure 13.2 Universal Event Log Dump - Execution from Windows Server .....	327
14	Databases .....	328
	Figure 14.1 Universal Spool List for Windows - List Universal Broker Database .....	344
	Figure 14.2 Universal Spool List for UNIX - List Universal Broker Database .....	344
	Figure 14.3 Universal Spool List for Windows - List Universal Event Monitor Spool Database Records .....	345
	Figure 14.4 Universal Spool List for UNIX - List Universal Event Monitor Spool Database Records .....	345
	Figure 14.5 Universal Spool List for Windows - List Broker Detail for a Component .....	346
	Figure 14.6 Universal Spool List for UNIX - List Broker Detail for a Component .....	346
	Figure 14.7 Universal Spool List for Windows - List Standard Out for a Component .....	347
	Figure 14.8 Universal Spool List for UNIX - List Standard Out for a Component .....	347
	Figure 14.9 Universal Spool Remove for Windows - Remove Component Records .....	349
	Figure 14.10 Universal Spool Remove for UNIX - Remove Component Records .....	349
	Figure 14.11 Universal Spool Remove for Windows - Remove Component Records .....	350
	Figure 14.12 Universal Spool Remove for UNIX - Remove Component Records .....	350



---

# List of Tables

---

Preface	27
Table P.1    Command Line Syntax	28
3    Remote Execution	69
Table 3.1    Remote Execution Primer Examples – exec Command Parameters	71
7    Security	164
Table 7.1    Certificate Map Matching Criteria	198
Table 7.2    Certificate Identifier Field	198
Table 7.3    Client IP Address - Matching Criteria	199
Table 7.4    Request Fields	201
Table 7.5    Certificate Fields	218
8    Configuration Management	224
Table 8.1    UNIX Configuration File Directory Search	232
Table 8.2    Stonebranch Solutions Configuration File Sample (infile.txt)	264
Table 8.3    Stonebranch Solutions Configuration File Sample (outfile.txt)	264
Table 8.4    Contents of outfile.txt after default merge	265
Table 8.5    Contents of outfile.txt when keeping unmatched destination values	266
Table 8.6    Contents of outfile.txt when using installation-dependent values	267
9    Component Management	268
Table 9.1    Universal Broker - Command Line Arguments to Daemon Startup Script	275
11   Message Translation	304
Table 11.1   Universal Message Translator – Translation Table	306
Table 11.2   Universal Message Translator for IBM i - Return Codes	306
15   Fault Tolerance Implementation	351
Table 15.1   Component Communication States	354

16	Network Data Transmission .....	355
	Table 16.1 Supported SSL cipher suites .....	357

---

# Preface

---

## Document Structure

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

## Cross-Reference Links

---

This document contains cross-reference links to and from other Stonebranch Solutions documentation.

In order for the links to work correctly:

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

## Conventions

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

### Typeface and Fonts

This document provides tables that identify how information is used. These tables identify values and/or rules that are either pre-defined or user-defined:

- *Italics* denotes user-supplied information.
- **Boldface** indicates pre-defined information.

Elsewhere in this document, **This Font** identifies specific names of different types of information, such as file names or directories (for example, `\abc\123\he1p.txt`).

### Command Line Syntax

Command line syntax, as shown in the examples throughout this document, use the following conventions:

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

Table P.1 Command Line Syntax

### Operating System-Specific Text

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

**z/OS**

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

This text resumes the information pertaining to all operating systems.

---

## Vendor References

References may be made in this document to a variety of vendor operating systems. We attempt to use the most current product names when referencing vendor software.

The following names may be used:

- **z/OS** is synonymous with IBM z/OS and IBM OS/390 line of operating systems.
- **Windows** is synonymous with Microsoft's Windows 2000 / 2003 / 2008, Windows XP, Windows Vista, and Windows 7 lines of operating systems. Any differences between the different systems will be noted.
- **UNIX** is synonymous with operating systems based on AT&T and BSD origins and the Linux operating system.
- **IBM i** is synonymous with IBM i/5, IBM OS/400, and OS/400 operating systems.
- **IBM System i** is synonymous with IBM i Power Systems, IBM iSeries, IBM AS/400, and AS/400 systems.

These names do not imply software support in any manner.

# Document Organization

This document provides information on how to use the Infitran business solution.

It is organized by Infitran feature, rather than by specific Infitran component. For example, Event Monitoring and File Triggering is a feature of Infitran; Universal Event Monitor is a component of Infitran.

Each chapter describes a separate feature and provides examples of how the feature can be implemented.

Each example is linked to detailed technical information about the Infitran component(s) that form the solution illustrated by that example.

- [Infitran Overview](#) (Chapter 1)  
Overview of the Infitran business solution.
- [Transferring Files to / from Remote Systems](#) (Chapter 2)  
Description and examples of the Transferring Files to / from Remote Systems feature.
- [Remote Execution](#) (Chapter 3)  
Description and examples of the Remote Execution feature.
- [Remote Execution for SAP Systems](#) (Chapter 4)  
Description and examples of the Remote Execution for SAP feature.
- [Web Services Execution](#) (Chapter 5)  
Description and examples of the Workload Execution feature.
- [Event Monitoring and File Triggering](#) (Chapter 6)  
Description and examples of the Event Monitoring and Triggers feature.
- [Security](#) (Chapter 7)  
Description and examples of the Security feature.
- [Configuration Management](#) (Chapter 8)  
Description and examples of the Configuration Management feature.
- [Component Management](#) (Chapter 9)  
Description and examples of the Component Management feature.
- [Messaging and Auditing](#) (Chapter 10)  
Description and examples of the Messaging and Auditing feature.
- [Message Translation](#) (Chapter 11)  
Description and examples of the Message Translation feature.
- [Monitoring and Alerting](#) (Chapter 12)  
Description and examples of the Monitoring and Alerting feature.
- [Windows Event Log Dump](#) (Chapter 13)  
Description and examples of the Windows Event Log Dump feature.
- [Databases](#) (Chapter 14)  
Description and examples of the Databases feature.
- [Fault Tolerance Implementation](#) (Chapter 15)  
Description and examples of the Fault Tolerance feature.
- [Network Data Transmission](#) (Chapter 16)  
Description and examples of the Network Data Transmission feature.
- [z/OS Cancel Command Support](#) (Chapter 17)  
Description and examples of the z/OS Cancel Command Support feature.

- [Glossary](#) (Appendix A)  
Glossary of terms used with Infitran.
- [Customer Support](#) (Appendix B)  
Customer support contact information for Infitran.

---

# Infitran Overview

---

## 1.1 What is Infitran?

Infitran (**I**ntelligent **F**ile **T**ransfer) is the Stonebranch Inc. business solution for Managed File Transfer.

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

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

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

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



## 1.2 What can Infitran do for Me?

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

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

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

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

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

## 1.3 Infitran Features

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

---

## 1.4 Infitran Components

**Infitran Features** are implemented via a set of inter-related components that provide for a complete intelligent file transfer business solution.

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

---

### Universal Data Mover

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

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

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

---

### Universal Event Monitor

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

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

---

## Universal Event Monitor for SOA

---

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

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

---

## Universal Enterprise Controller

---

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

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

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

---

## Universal Event Subsystem

---

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

---

## Universal Enterprise Controller Client Applications

---

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

### I-Activity Monitor

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

### I-Management Console

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

### I-Administrator

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

---

## Universal Broker

---

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

---

## Stonebranch Solutions Utilities

---

**Stonebranch Solutions Utilities**, included as part of the Infitran business solution, perform a variety of functions for one or more operating systems.

### Universal Certificate

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

### Universal Control

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

### Universal Database Dump

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

### Universal Database Load

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

### Universal Display Log File

Universal Display Log File is available on the iSeries platform. It provides the ability to read job log files and write them to standard out and optionally delete the files after read.

### Universal Encrypt

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

## Universal Event Log Dump

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

## Universal Message Translator

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

## Universal Install Merge

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

## Universal Query

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

## Universal Return Code

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

## Universal Spool List

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

## Universal Spool Remove

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



---

## Universal Submit Job

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

## Universal Write-to-Operator

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

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

---

## 1.5 Limited Use Components

The following Indesca business solution components are included in Infitran with limited use through the [exec](#) and [execsap](#) commands only.

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

---

### Universal Command

---

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

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

#### Limitations of Use

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

---

### Universal Command Agent for SOA

---

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

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

#### Limitations of Use

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

---

## Universal Connector

---

**Universal Connector** is a command line interface that lets you manage SAP background processing tasks from any scheduling system on any platform.

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

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

### Limitations of Use

If **Universal Connector** is on the same system as **Universal Data Mover**, you can execute SAP events using the Universal Data Mover `execsap` command.

# Transferring Files to / from Remote Systems

---

## 2.1 Overview

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

This chapter describes the framework in which transfers are made, and provides examples of file transfers from all supported operating systems.

---

## 2.2 Transfer Operation Components

There are three components to any Infitran transfer operation:

1. Manager
2. Primary server
3. Secondary server

The Manager can act as the primary server, depending on the type of transfer session: two-party or three-party (see Section [2.3 Transfer Sessions](#)).

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

---

### 2.2.1 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.

---

### 2.2.2 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 Section [2.3.3 Three-Party Transfer Sessions](#)).

---

### 2.2.3 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.

---

## 2.3 Transfer Sessions

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. 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.

---

### 2.3.1 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.

---

### 2.3.2 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 [Figure 2.1 Infitran Transfer Sessions.](#))

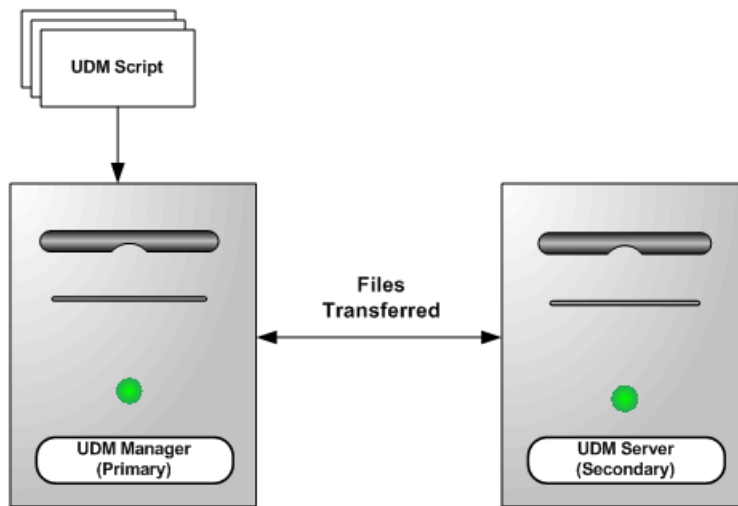
---

### 2.3.3 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 [Figure 2.1 Infitran Transfer Sessions.](#))

### Two-Party Transfer



### Three-Party Transfer

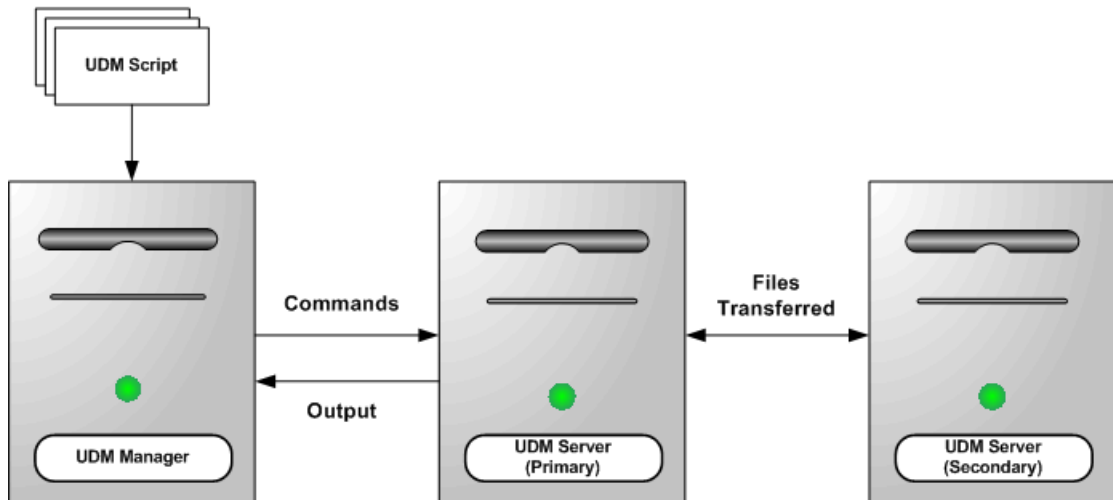


Figure 2.1 Infitran Transfer Sessions

---

## 2.4 Transferring Files Examples

This section provides examples, specific to the operating systems supported by Stonebranch Solutions, for the Transferring Files to / from Remote Systems feature of Infitran.

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

### z/OS

---

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.

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

[Copy a Set of Files to a New z/OS Partitioned Data Set](#)

**Note:** These z/OS examples apply equally as well to the Windows and UNIX operating systems, with appropriate changes for the file system syntactical differences.

### UNIX and Windows

---

These examples illustrate two-party transfer sessions.

[Simple File Copy to the Manager](#)

[Simple File Copy to the Server](#)

[Copy a Set of Files](#)

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.)



## IBM i

---

These examples illustrate two-party transfer sessions between IBM i and UNIX.

Each example illustrates a file transfer for the LIB file system.

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 the list of [UNIX and Windows](#) examples.

[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 a Batch Job](#)

**Note:** These examples apply equally as well to the Windows and UNIX operating systems, with appropriate changes for the file system syntactical differences.

## 2.4.1 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=so19 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 printed out 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 `so19`. The host `so19` 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 `so19`.
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=so19 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 the line 6, which sets the local attribute `createop`.  
The `createop` attribute 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](#)

## 2.4.2 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=so19 user=top098 pwd=p100m
4 filesystem 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=so19 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

## 2.4.3 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 _halt=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](#)

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

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

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

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

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

### DSN File System

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

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

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

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

### Components

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

## 2.4.5 Copy a Set of Files to a New z/OS 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](#)



## 2.4.6 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=so19 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 printed out 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 `so19`. The host `so19` 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 `so19`.
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](#)

## 2.4.7 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=so19 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 `so19`. The host `so19` 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 `so19`.
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](#)

## 2.4.8 Copy a Set of Files

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

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 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](#)

## 2.4.9 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 filesystem 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**.

The **createop** attribute 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](#)

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

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

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

### LIB File System

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

### Components

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

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

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

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

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

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

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

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

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

### LIB File System

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

### Components

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

## 2.4.12 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](#)



## 2.4.13 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=so19 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](#)

## 2.4.14 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](#)

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

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

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

### LIB File System

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

### Components

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

## 2.4.16 Invoke a Script from a 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](#)

# Remote Execution

## 3.1 Overview

This chapter provides information on the Remote Processing features and functionality of the Infitran business solution.

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

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

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

Infitran Remote Execution using Universal Command consists primarily of two Stonebranch Solutions 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.

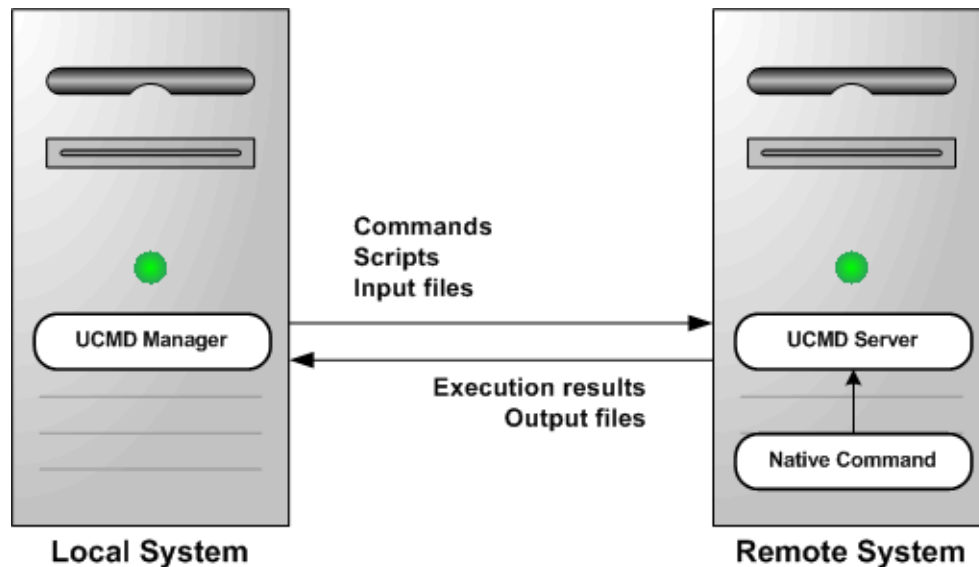


Figure 3.1 Remote Execution Components: UCMD Manager and Server

## 3.2 Execution Primer

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

**Note:** Please read Section [3.1 Overview](#) prior to reading this section, which builds upon the material presented in the Overview.

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

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

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

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 <code>joe</code> . 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 <code>abcdefg</code> . This will need to be changed to the password for the USER_ID you use to execute the remote command.

Table 3.1 Remote Execution Primer Examples – exec Command Parameters

## 3.2.1 Remote Execution Requirements

---

This section 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 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 Reference contains platform-specific information for invoking UCMD Manager. A UCMD Server installed on the remote system receives the command specified by `exec`'s `cmd` parameter and executes it.

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

For information on securing access to Indesca components, see Chapter 6 [Security](#) in the [Indesca User Guide](#).

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 `da11as` 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 (`da11as`). 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.



**z/OS**

This section illustrates 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 [Figure 3.2](#); otherwise, use something similar to [Figure 3.3](#). To execute a command on a remote system using an active UDM transfer session, follow the example shown in [Figure 3.4](#).

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

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

Figure 3.2 Remote z/OS Execution Using an IP Address

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

Figure 3.3 Remote z/OS Execution Using a Host Name

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

Figure 3.4 Remote z/OS Execution Using a UDM Logical Session Name

## Windows

This section illustrates how to execute a process on a remote Windows system using the UDM [14.18 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 [Figure 3.5](#); otherwise, use something similar to [Figure 3.6](#).

To execute a command on a remote system using an active UDM transfer session, follow the example shown in [Figure 3.7](#).

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

Figure 3.5 Remote Windows Execution Using an IP Address

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

Figure 3.6 Remote Windows Execution Using a Host Name

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

Figure 3.7 Remote Windows Execution Using a UDM Logical Session Name

**UNIX**

This section illustrates 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 [Figure 3.8](#); otherwise, use something similar to [Figure 3.9](#).

To execute a command on a remote system using an active UDM transfer session, follow the example shown in [Figure 3.10](#).

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

Figure 3.8 Remote UNIX Execution Using an IP Address

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

Figure 3.9 Remote UNIX Execution Using a Host Name

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

Figure 3.10 Remote UNIX Execution Using a UDM Logical Session Name

**IBM i**

This section illustrates 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 [Figure 3.11](#); otherwise, use something similar to [Figure 3.12](#). To execute a command on a remote system using an active UDM transfer session, follow the example shown in [Figure 3.13](#).

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.

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

Figure 3.11 Remote IBM i Execution Using an IP Address

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

Figure 3.12 Remote IBM i Execution Using a Host Name

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

Figure 3.13 Remote IBM i Execution Using a UDM Logical Session Name

---

## 3.3 Remote Execution Examples

This section provides examples, specific to the operating systems supported by Stonebranch Solutions, for the Remote Execution feature of Infitran.

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

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

---

### Windows

[Windows Directory Listing Using a Batch File - Default Directory](#)

[Windows Directory Listing Using a Batch File - Returned File](#)

---

### UNIX

[UNIX Listing Using a Shell Script](#)

[UNIX - Integrating UDM with FTP Using a Shell Script](#)

[UNIX - Integrating UDM with FTP Using a Command Reference](#)

---

### IBM i

[IBM i from Windows, UNIX, or IBM i - exec Command Return Codes](#)

### 3.3.1 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
```

Figure 3.14 exec Command Under Windows - Listing in Default Directory

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

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

Figure 3.15 Listing in Default Directory - Example Batch File

Output sent to `stdout.txt`.

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

Directory of C:\wrk\xmp\win

07/27/2010  03:27 PM    <DIR>          .
07/27/2010  03:27 PM    <DIR>          ..
07/27/2010  10:08 AM                20 winxmp.bat
07/27/2010  03:46 PM            106 winxmpbat.udm
                2 File(s)        126 bytes
                2 Dir(s)  13,453,979,648 bytes free
```

Figure 3.16 Listing in Default Directory - Listing Sent to stdout.txt

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

```

2010.07.27 16.06.47.541 UNV2800I Universal Data Mover 4.2.0 Level 1 Release
Build 105 started.
2010.07.27 16.06.47.556 Processing script: winxmpbat.udm
2010.07.27 16.06.47.556 exec 192.168.20.47 cmd="C:\wrk\xmp\win\winxmp.bat >
stdout.txt" user=joe pwd=*
2010.07.27 16.06.48.431 quit
2010.07.27 16.06.48.447 Finished processing script: winxmpbat.udm
2010.07.27 16.06.49.447 UNV2801I Universal Data Mover 4.2.0 Level 1 Release
Build 105 ended successfully.

```

Figure 3.17 Listing in Default Directory - Transaction log from UDM via stdout

## 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](#)

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

This example builds on the example illustrated in Section [3.3.1 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. Due to the complexity of this example, each line will be explained. Each line is numbered for your convenience.

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

```
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
```

Figure 3.18 exec Command Under Windows - Returned File



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
```

Figure 3.19 Returned File - Example Batch File

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

07/27/2010  03:27 PM    <DIR>          .
07/27/2010  03:27 PM    <DIR>          ..
07/27/2010  10:08 AM                20 winxmp.bat
07/27/2010  03:46 PM               106 winxmpbat.udm
                2 File(s)                126 bytes
                2 Dir(s)  13,453,979,648 bytes free
```

Figure 3.20 Returned File - Listing Sent to stdout.txt

## UDM exec Command Parameters

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

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

## Components

[Universal Data Mover Manager for Windows](#)

[Universal Command Server for Windows](#)

### 3.3.3 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.

Keep in mind that 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.

#### 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.

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

Figure 3.21 UNIX Listing - UDM Script on Local System

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
```

Figure 3.22 UNIX Listing - Shell Script on Remote System

Output sent to `stdout.txt`.

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

Figure 3.23 UNIX Listing - Listing Sent to stdout.txt

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

```
2010.07.28 10.13.06.845 UNV2800I Universal Data Mover 4.2.0 Level 0 Release
Build 104 started.
2010.07.28 10.13.06.845 Processing script: ls.udm
2010.07.28 10.13.06.847 open r=houston user=joe pwd=* port=7887
2010.07.28 10.13.07.114 Data session established using cipher: NULL-MD5
2010.07.28 10.13.07.159 Two party session established with r (component
1278600806)
2010.07.28 10.13.07.161 Transfer mode settings:
2010.07.28 10.13.07.198 type=binary
2010.07.28 10.13.07.198 trim=no
2010.07.28 10.13.07.198 Session options:
2010.07.28 10.13.07.198 Keep Alive Interval: 120
2010.07.28 10.13.07.198 Network Fault Tolerant: yes
2010.07.28 10.13.07.198 exec r cmd="sh /home/joe/wrk/xmp/ls/ls.sh" user=joe
pwd=* port=7887
ls.sh
stdout.txt
2010.07.28 10.13.08.072 quit
2010.07.28 10.13.08.074 Session closed
2010.07.28 10.13.08.074 Finished processing script: ls.udm
2010.07.28 10.13.10.074 UNV2801I Universal Data Mover 4.2.0 Level 0 Release
Build 104 ended successfully.
```

Figure 3.24 UNIX Listing - UDM Manager Transaction Log

## 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 <a href="#">exec</a> command

## Components

[Universal Data Mover Manager for UNIX](#)

[Universal Command Server for UNIX](#)

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

---

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

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

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

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

#### 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 `loca1` to the system on which the UDM Manager is running. Setting up `loca1` 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.

```

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

```

Figure 3.25 Integrating UDM with FTP - UDM Script on Local System

The shell script sets up and executes FTP commands.

```

ftp -ipnv houston <<FTP_DONE
quote USER joe
quote PASS abcdefg
cd /home/joe/tmp
lcd /home/joe/wrk/xmp/dmzFtp
put file.txt file.new.txt
quit
FTP_DONE
exit 0

```

Figure 3.26 Integrating UDM with FTP - Shell Script on Remote System

## UDM exec Command Parameters

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

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

## Components

[Universal Data Mover Manager for UNIX](#)

[Universal Command Server for UNIX](#)

## 3.3.5 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.

This example is based on the example in Section [3.3.4 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 the Command References section in Chapter [10 Universal Command Server for UNIX](#) of the Universal Command [4.2.0 Reference Guide](#).

### 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 UCMD Server option `CMD_REFERENCE_DIRECTORY`. 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
```

Figure 3.27 Using a Command Reference - UDM Script on Local System

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
<eof>

ftp -ipnv $1 <<FTP_DONE
quote USER joe
quote PASS abcdefg
cd /home/amos/tmp
lcd /home/joe/wrk/xmp/dmzFtp
put file.txt file.new.txt
quit
FTP_DONE
exit 0
```

Figure 3.28 Using a Command Reference - Command Reference on Remote System

## UDM exec Command Parameters

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

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



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

---

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

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.

### 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.

```
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
```

Figure 3.29 UDM Script - Exec Command Return Codes

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

## Components

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

[Universal Data Mover Manager for Windows](#)

[Universal Data Mover Manager for UNIX](#)

[Universal Command Manager for IBM i](#)

# Remote Execution for SAP Systems

---

## 4.1 Overview

This chapter provides information on the Remote Processing for SAP feature and functionality of the Infitran business solution.

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

---

## 4.2 Remote Execution of SAP Examples

This section provides examples of the Remote Execution of SAP feature of Infitran using the Universal Data Mover [execsap](#) command.

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

### z/OS

---

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

### UNIX

---

[Raising an SAP Event for UNIX Example](#)

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

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

In this example (Figure 4.1 and Figure 4.2, below), we assume the following scenario:

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

Infitran is being run on a z/OS system to transfer three data files (`cars.dat`, `trucks.dat`, and `boats.dat`) from remote system `so19` 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.

```
//UDMEXSAP JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*****
//* Description
//* -----
//* This sample opens a three-party transfer session between hosts
//* so19 and SAP001. Three files are transferred from so19 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=so19 dest=SAP001 xfile=xuser1

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

Figure 4.1 Raising an SAP Event for z/OS - JCL (1 of 2)

```
*****
#* 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
/*
```

Figure 4.2 Raising an SAP Event for z/OS - JCL (2 of 2)

## Components

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

[Universal Data Mover Server for UNIX](#)

[Universal Connector for z/OS](#)

## 4.2.2 Raising an SAP Event for UNIX Example

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

In this example (Figure 4.3 and Figure 4.4, below), we assume the following scenario:

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

Infitran is being run on a UNIX system to transfer three data files (`cars.dat`, `trucks.dat`, and `boats.dat`) from remote system `so19` 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.

```
*****  
**  
# Description  
# -----  
# This sample opens a three-party transfer session between hosts  
# so19 and SAP001. Three files are transferred from so19 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=so19 dest=SAP001 xfile=xuser1  
attrib dest createop=replace  
cd src=/opt/app/data  
cd dest=/input
```

Figure 4.3 Raising an SAP Event for UNIX - UDM Script File: BIVehicle001 (1 of 2)



```
*****
#* 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
```

Figure 4.4 Raising an SAP Event for UNIX - UDM Script File: BIVehicle001 (2 of 2)

## Components

[Universal Data Mover Manager for UNIX](#)

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

[Universal Connector for z/OS](#)

# Web Services Execution

---

## 5.1 Overview

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

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

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

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

---

## 5.2 Web Services Examples

This chapter provides examples for the Web Services Execution feature of Infitran.

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

### UNIX

---

[Inbound JMS Examples](#)

[Inbound SOAP Implementation](#)

## 5.2.1 Inbound JMS Examples

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`.

Figure 5.1, below, 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>
```

Figure 5.1 Inbound JMS - Constructing Connection to Target

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

Figure 5.2, below, 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>

```

Figure 5.2 Inbound JMS - Attachment to an Apache ActiveMQ Dynamic Topic

## Websphere Queue

Figure 5.3, below, 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>

```

Figure 5.3 Inbound JMS - Attachment to an IBM Websphere Queue

## MQ Series Queue:

Figure 5.4, below, 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:MQLListener>
      <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:MQLListener>
  </sb:Listeners>
</sb:MQConnection>
```

Figure 5.4 Inbound JMS - Attachment to an IBM MQ Series Queue

## Triggering an Event

Once a file has been written to the filesystem, UEM could be used to trigger an event (see [Figure 5.5](#), below).

This event 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 a `xml` extension. It then executes the handler, which runs UDM with a script.

The UDM script looks for all files that begin with a `2` and end with `.xml` on the local server. These files 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
```

Figure 5.5 Triggering an Event

## 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](#)

[Universal Event Monitor for SOA](#)

## 5.2.2 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/universa1/UAC.xml` file.

```
<?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>>
```

Figure 5.6 Inbound SOAP Request UAC.xml

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.

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:identitySourceApplication
Id>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>OD
PT0001</NS1:activityAction><NS1:activityStartDate>2010-02-24</NS1:activity
StartDate><NS1:activityStartTime>08:35:42.397382</NS1:activityStartTime></NS1:
:process></soapenv:Body></soapenv:Envelope>
```

Figure 5.7 Inbound SOAP Request – Message Payload Written to `process_%Seq%.xml` File

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

- **identitySourceApplicationId**
- **activityRequestId**
- **activityAction**

Figure 5.8, below, 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
```

Figure 5.8 Inbound SOAP Request – Universal Event Monitor 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
```

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

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

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
```

Figure 5.9 Inbound SOAP Request – Universal Event Monitor Handler Definition

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/uac1.conf`:

- `uem_handler control,allow,noauth`

Changes to the configuration files require the Universal Broker to be refreshed (see Section 8.5 [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.

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

```

Figure 5.10 Outbound SOAP Request – abc.rexx

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 Section [8.5 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\_%.xml is processed. This file is useful as an audit trail and for problem diagnosis.

In order to ensure that this file does not grow to an unreasonable size, an 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"
ACTION_TYPE   CMD
MAXRC         0
USERID        "control"
PWD           "UACL"
CMD           "rm /export/home/control/indesca/abc.log"
END_HANDLER

```

Figure 5.11 Outbound SOAP Request – Event and Handler to purge abc.log

## Components

[Universal Event Monitor](#)

[Universal Event Monitor for SOA](#)

[Universal Broker](#)

[Universal Write-to-Operator](#)

[Universal Encrypt](#)

# Event Monitoring and File Triggering

---

## 6.1 Overview

The Event Monitoring and File Triggering feature of Infitran provides a consistent, platform-independent means of monitoring one or more local or remote system events. It also can execute a system command or script based on the outcome of the events that it monitors.

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

The methods available for defining an event and its associated actions are described in the following sections.

## 6.2 Universal Event Monitor

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

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

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

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

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

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



Figure 6.1, below, illustrates the interaction of the various components that make up Universal Event Monitor.

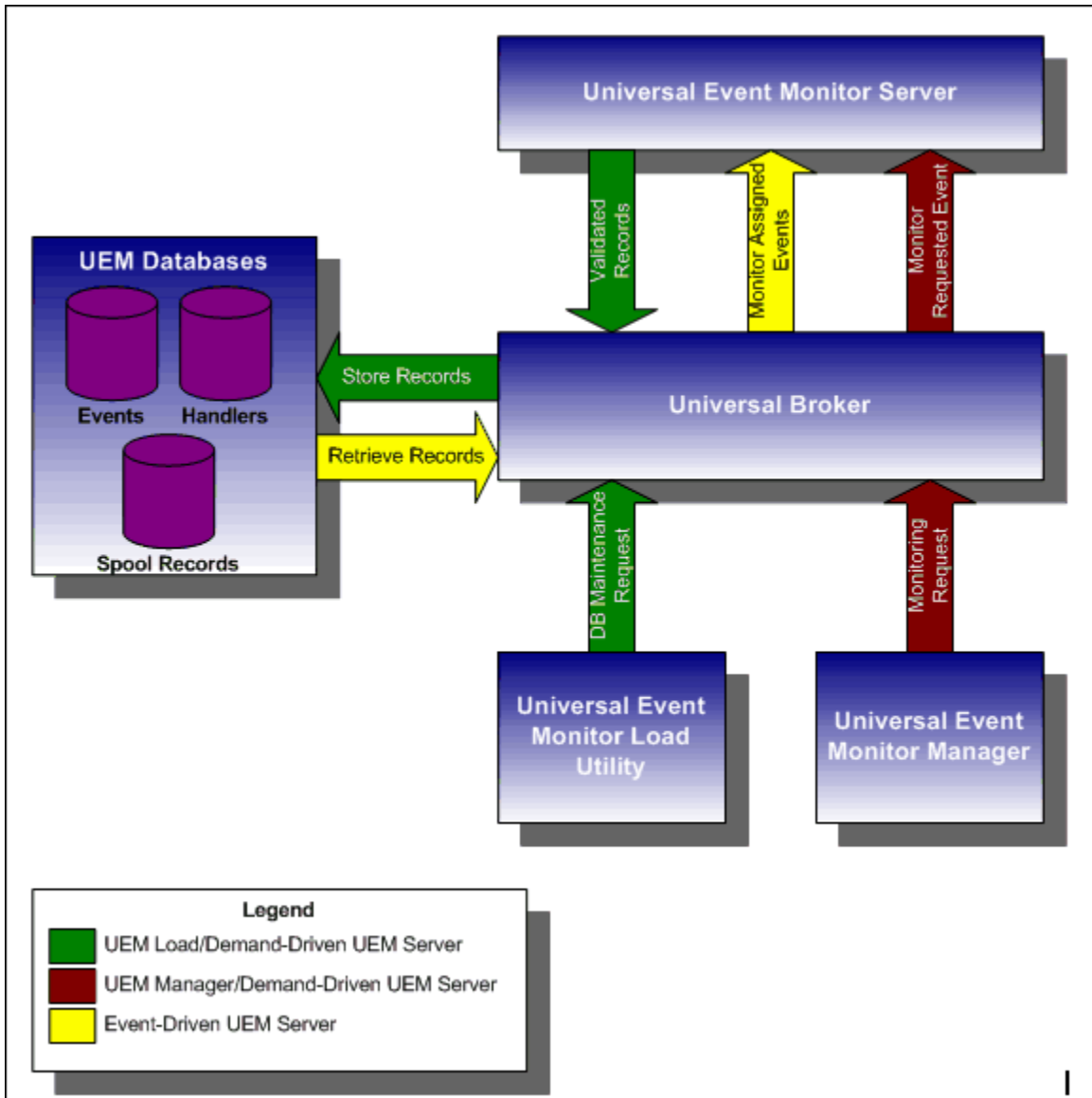


Figure 6.1 High-Level Interaction of UEM Components

---

## 6.2.1 Storing Event Definitions and Event Handlers

---

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

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

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

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

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

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

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

Figure 6.2, below, illustrates the interaction of the Universal Broker and the Universal Event Monitor Server components involved during the execution of UEMLoad.

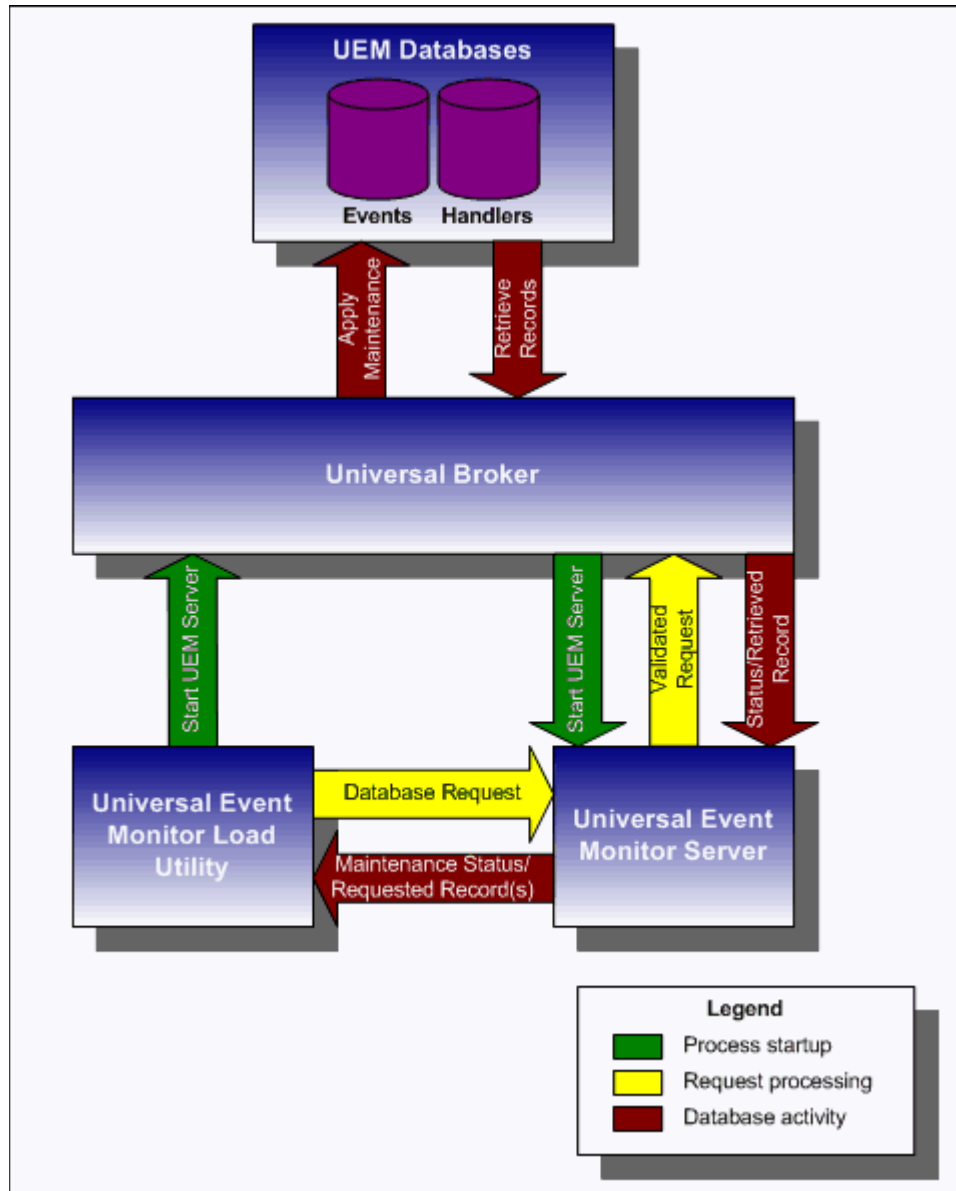


Figure 6.2 UEMLoad Utility Overview

---

## 6.2.2 Monitoring a Single Event

---

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

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

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

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

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

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

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

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

Figure 6.3, below, illustrates the interaction of the Universal Broker and the Universal Event Monitor components involved when a UEM Manager is executed.

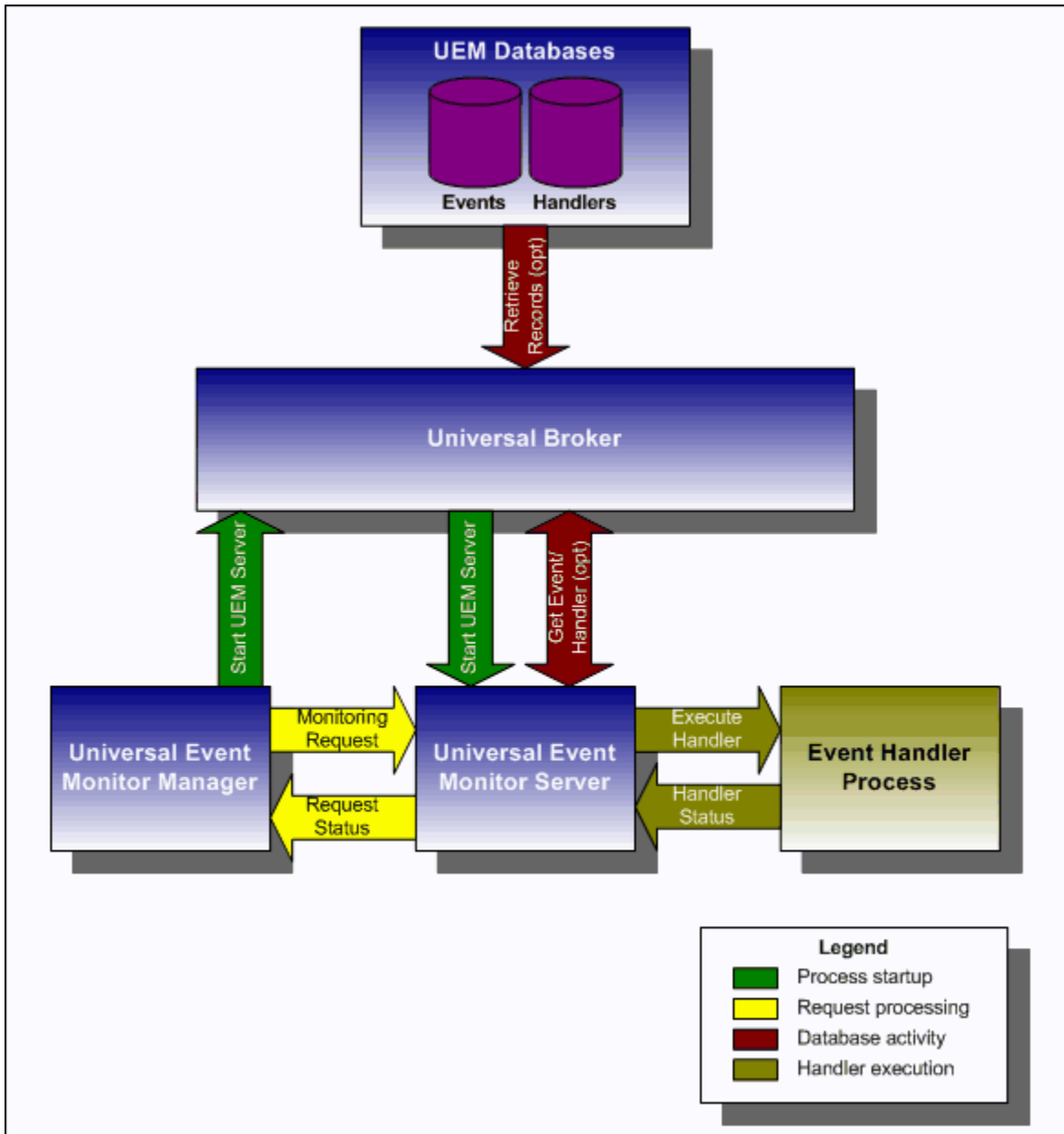


Figure 6.3 UEM Manager Overview

---

## 6.2.3 Monitoring Multiple Events

---

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

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

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

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

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

Figure 6.4, below, illustrates the interaction of the Universal Broker and an event-driven UEM Server.

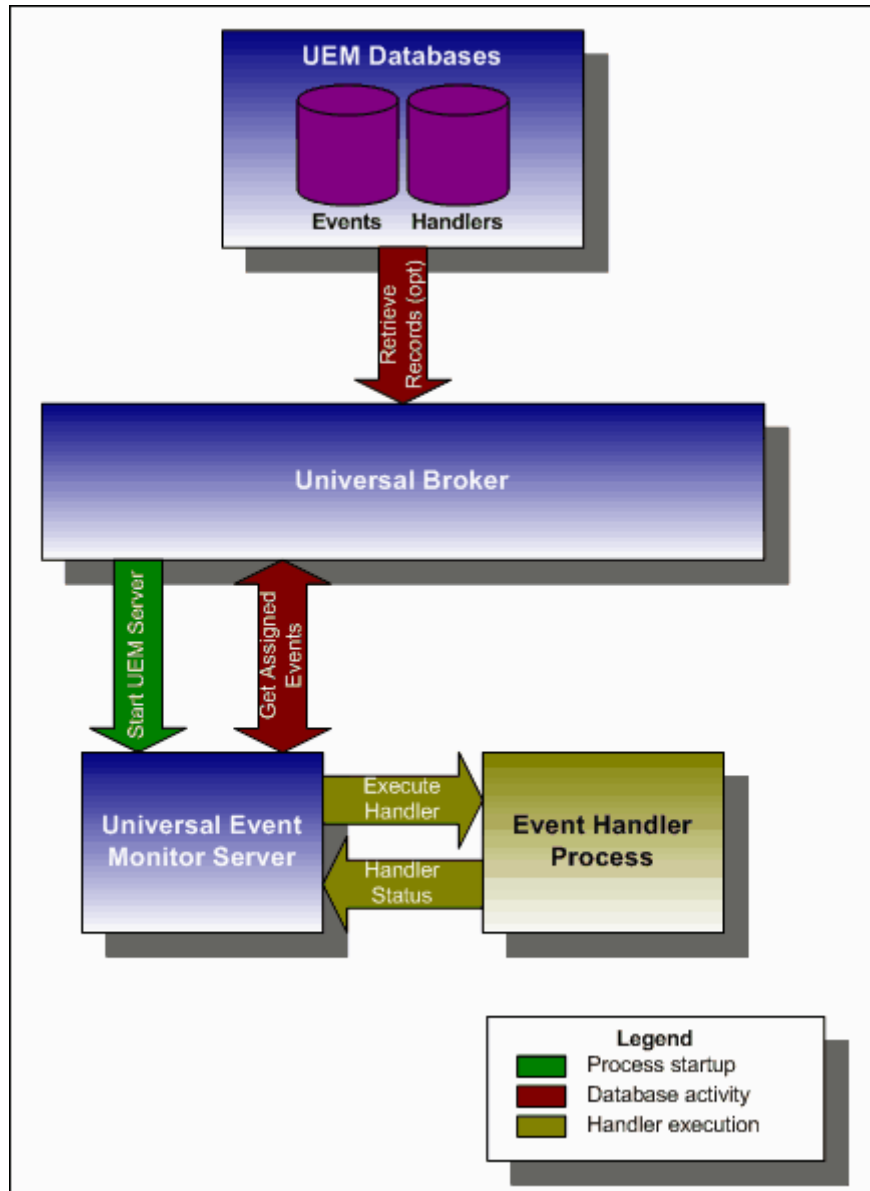


Figure 6.4 UEM Server Overview

## 6.3 UEMLoad

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

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

The UEMLoad utility (**uemload**) manages the event definition and event handler database files. (For information on the spool database file, see [Chapter 7 Universal Event Monitor Server](#) in the [Universal Event Monitor Reference Guide](#).)

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 [Chapter 13 UEMLoad Utility](#) in the [Universal Event Monitor Reference Guide](#).



---

## 6.3.1 Controlling Database Access

---

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

However, there are utilities provided, including Universal Spool List (`uslist`) and Universal Spool Remove (`us1rm`) 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 [Stonebranch Solutions Utilities 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 Section [13.2.1 Database Files Location](#).

### 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.

## 6.4 Event Monitoring and File Triggering Examples

This section provides examples, specific to the operating systems supported by Stonebranch Solutions, for the Event Monitoring and File Triggering feature of Infitran.

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

### Universal Event Monitoring Examples

The examples utilizing Universal Event Monitor assume the following information:

- UEM Server is installed on a remote system named **uemhost**.
- Security option has been enabled in the UEM Server's configuration.

The values for the **-userid** and **-pwd** parameters represent the user ID and password of a valid user account defined on **uemhost**.

### z/OS

---

[Starting an Event-Driven Server](#)

[Refreshing an Event-Driven UEM Server](#)

[Using a Stored Event Handler Record in z/OS](#)

[Handling an Event With a Script in z/OS](#)

[Handling an Expired Event in z/OS](#)

[Continuation Character - in z/OS Handler Script](#)

[Continuation Character + in z/OS Handler Script](#)

[Continuation Characters - and + in z/OS Handler Script](#)

### Windows

---

[Using a Stored Event Handler Record in Windows](#)

[Executing a Script for a Triggered Event Occurrence in Windows](#)

[Handling an Expired Event in Windows](#)

[Adding a Single Event Record for Windows](#)

[Adding a Single Event Handler Record for Windows](#)

---

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

## UNIX

---

- [Using a Stored Event Handler Record in UNIX](#)
- [Executing a Script for a Triggered Event Occurrence in UNIX](#)
- [Handling an Expired Event in UNIX](#)
- [Adding a Single Event Record for UNIX](#)
- [Adding a Single Event Handler Record for UNIX](#)
- [Listing All Event Definitions for UNIX](#)
- [Exporting the Event Definition and Event Handler Databases for UNIX](#)
- [List a Single Event Handler Record for UNIX](#)
- [Listing Multiple Event Definitions and Event Handlers Using Wildcards for UNIX](#)
- [Add Record\(s\) Using a Definition File for UNIX](#)
- [Add Record\(s\) Remotely, Using a Definition File Redirected from STDIN for Windows](#)
- [Add Record\(s\) Remotely, Using a Definition File Redirected from STDIN \(for z/OS\) for Windows](#)
- [Definition File Format for UNIX](#)

## 6.4.1 Starting an Event-Driven Server

There are two ways start a UEM event-driven 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
/*
```

Figure 6.5 Starting a UEM Event-Driven Server

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

### Components

[Universal Control](#)

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

## 6.4.2 Refreshing an Event-Driven UEM Server

In this example, RESUEMS will refresh the UEM event-driven 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
/*
```

Figure 6.6 Refreshing a UEM Event-Driven Server

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

### Components

[Universal Control](#)

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

### 6.4.3 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_id h001
-host uemhost
-userid uemuser
-pwd uemusers_password
-max_count 1
/*
```

Figure 6.7 Using a Stored Event Record

## Components

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

[Universal Event Monitor Server](#)

---

## 6.4.4 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
/*

```

Figure 6.8 Handling an Event with a Script

## Components

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

[Universal Event Monitor Server for Windows](#)

## 6.4.5 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 -a1R /home` command.

Note that the `-expired` option also contains the `-options` parameter. In this example, the `-options` parameter redirects the output of the `ls -a1R /home` command to a file in uemuser's home directory, `uemtest.log`.

```
//jobname JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//STEP1 EXEC UEMPRC
//SYSIN DD *
-event_type file
-filespec uemtest.dat
-inact_date_time +1
-expired -cmd "ls -a1R /home" -options ">uemtest.log 2>&1"
-host uemhost
-userid uemuser
-pwd uemusers_password
/*
```

Figure 6.9 Handling an Expired Event

### Components

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

[Universal Event Monitor Server for UNIX](#)

## 6.4.6 Continuation Character - in z/OS Handler Script

Continuation characters ("- and "+") are useful when you want to execute a script line that is longer than your available z/OS character space.

The "-" continuation character will preserve trailing spaces in your line.

The "+" continuation character will not preserve trailing spaces in your line.

**The following z/OS handler script:**

```
begin_script
  stmt "ls -a -          <---- Notice the continuation character "-"
  >dirfile"
end_script
```

**Will produce the following output when loaded to the uemh.db:**

```
Handler ID.....: MFCTRIGGER_1
Handler Type.....: SCRIPT
Max Acceptable Return Code...: 0
Encrypted User File.....:
User ID.....: mfc1a
Script statements:
  ls -a >dirfile
  Script Type.....: bat
Command Line Options.....:
Last Modified On.....: 06/11/09 10:32:31 AM
Last Modified By.....: mfc1a
```

Figure 6.10 Continuation Character "-" in z/OS Handler Script

## Components

### Universal Event Monitor Manager for z/OS

## 6.4.7 Continuation Character + in z/OS Handler Script

Continuation characters ("- and "+) are useful when you want to execute a script line that is longer than your available z/OS character space.

The "-" continuation character will preserve trailing spaces in your line.

The "+" continuation character will not preserve trailing spaces in your line.

**The following z/OS handler script:**

```
begin_script
  stmt "ls -a >dir +          <---- Notice the continuation character "+"
  file"
end_script
```

**Will produce the following output when loaded to the uemh.db:**

```
Handler ID.....: MFCTRIGGER_1
Handler Type.....: SCRIPT
Max Acceptable Return Code...: 0
Encrypted User File.....:
User ID.....: mfc1a
Script statements:
  ls -a >dirfile
Script Type.....: bat
Command Line Options.....:
Last Modified On.....: 06/11/09 11:46:32 AM
Last Modified By.....: mfc1a
```

Figure 6.11 Continuation Character "+" in z/OS Handler Script

## Components

### Universal Event Monitor Manager for z/OS

## 6.4.8 Continuation Characters - and + in z/OS Handler Script

Continuation characters ("- and "+) are useful when you want to execute a script line that is longer than your available z/OS character space. The "-" continuation character will preserve trailing spaces in your line. The "+" continuation character will not preserve trailing spaces in your line.

This example shows the use of "+" to concatenate a command line or a word within a z/OS script without a space as the use of "-" to continue a line of script where a space is required within the same z/OS handler script.

### The following z/OS handler script:

```
begin_script
stmt "ls -a >dir +
file"
stmt "uemFName=`basename \u201c$UEMORIGFILE\u201d | sed 's/\(.*\) +
.\(.*\) / \1/'`"
stmt "fname=$uemFName.$dt.$tm.$pid.txt"
stmt " ls -al >dir+
data"
stmt "ls -a -
>new+
data"
end_script
```

### Will produce the following output when loaded to the uemh.db:

```
Handler ID.....: MFCTRIGGER_1
Handler Type.....: SCRIPT
Max Acceptable Return Code...: 0
Encrypted User File.....:
User ID.....: mfc1a
Script statements:
ls -a >dirfile
uemFName=`basename \u201c$UEMORIGFILE\u201d | sed 's/\(.*\) +
.\(.*\) / \1/'`
fname=$uemFName.$dt.$tm.$pid.txt
ls -al >dirdata
ls -a >newdata
Script Type.....: bat
Command Line Options.....:
Last Modified On.....: 06/11/09 01:25:20 PM
Last Modified By.....: mfc1a
```

Figure 6.12 Continuation Characters "-" and "+" in z/OS Handler Script

## Components

### Universal Event Monitor Manager for z/OS

## 6.4.9 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
```

Figure 6.13 Using a Stored Event Handler Record

### Components

#### Universal Event Monitor Manager for Windows

## 6.4.10 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"
```

Figure 6.14 Handling an Event with a Script.

Figure 6.15, below, illustrates the contents of the `C:\UEMScripts\h_001.txt` file.

```
#!/bin/sh

# Sample script h_001.txt

argNum=1

# Display each command line argument.
while [ "$1" != "" ]
do
echo Parm $argNum: $1
shift
argNum=`expr $argNum + 1`
done
```

Figure 6.15 Contents of Sample Script File

## Components

[Universal Event Monitor Manager for Windows](#)

[Universal Event Monitor Server for UNIX](#)



## 6.4.1.1 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` parameter. This is necessary to accommodate parsing requirements for command options in Windows (see the [FILE\\_SPECIFICATION](#) option in the [Universal Event Monitor Reference Guide](#)).

A relative inactive date / time instructs the demand-driven Server to monitor the event for one (1) minute. If the UEM Server detects the file, and the file completes within that time, UEM sets the event occurrence to the triggered state. Since the command options contain no event handler information for a triggered occurrence, the UEM Server simply renames the file (by default). UEM Server then makes the event inactive, and ends.

If the UEM Server detects the file, but the file does not complete within 60 seconds, UEM sets the occurrence to the rejected state. Since the command options contain no event handler information for a rejected occurrence, the UEM Server leaves the file as-is and takes no further action.

If the UEM Server does not detect the presence of `uemtest.dat` before the inactive time elapses, it sets the event to the expired state. When this happens, the UEM Server executes the command specified by the `-cmd` parameter of the `-expired` option. In this example, UEM executes the `'ls -a1R /uem files'` command.

Note that the `-expired` option also contains the `-options` parameter. In this example, the `-options` parameter redirects the output of the `'ls -a1R /uem files'` command to a file in `uemuser's` home directory called `uemtest.log`.

```
uem -host uemhost -event_type file
-userid uemuser -pwd uemusers_password
-filespec " /uem files/uemtest.dat"
-inact_date_time +1
-expired -cmd "ls -a1R '/uem files'" -options ">uemtest.log 2>&1"
```

Figure 6.16 Handling an Expired Event

### Components

[Universal Event Monitor Manager for Windows](#)

[Universal Event Monitor Server for UNIX](#)

## 6.4.12 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 `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
```

Figure 6.17 Adding a single event definition record.

### Components

[UEMLoad Utility for Windows](#)

[Universal Event Monitor Server for UNIX](#)

## 6.4.13 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"
```

Figure 6.18 Adding a single event handler record.

### Components

[UEMLoad Utility for Windows](#)

[Universal Event Monitor Server for UNIX](#)

[Universal Encrypt](#)

## 6.4.14 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 *
```

Figure 6.19 Listing all event definition records.

**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.

### Components

#### [UEMLoad Utility for Windows](#)

## 6.4.15 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 examples shown in [Figure 6.27](#).

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

Figure 6.20 Exporting all Event and Handler Records

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

### Components

[UEMLoad Utility for Windows](#)

## 6.4.16 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
```

Figure 6.21 List a Single Event Handler Record

Figure 6.22, below, illustrates sample output for this command. (The values shown are those that could be expected if the record were added using the command shown in Figure 6.17.)

In this specific instance, the user ID contained in `encrypted.file` (from Figure 6.17) is `sparkie`, and the record was added by the user account with an ID of `sbuser`.

```
UNV3659I Connecting to local broker on port 7887.
UNV3406I Universal Event Monitor Server component 1117035117 started.
UNV3666I Load request started at 11:32:45 AM 05/25/2005.

Event Handler(s):
=====
Handler ID.....: dirlist
Max Acceptable Return Code.: 0
User ID.....: sparkie
Command.....: ls -al
Last Modified On.....: 05/25/2005 11:32:06 AM
Last Modified By.....: sbuser

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

Figure 6.22 Sample List Output

## Components

### UEMLoad Utility for Windows

## 6.4.17 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??
```

Figure 6.23 Using Wildcards to List Records

### Components

[UEMLoad Utility for Windows](#)

## 6.4.18 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 [Figure 6.27](#).

A definition file allows multiple records to be added to the event definition and/or event handler databases at the same time. When no definition file is used, only a single record can be added to the database(s).

```
uemload -add -deffile uemadd.txt
```

Figure 6.24 Add Database Record(s) Using a Definition File

### Components

[UEMLoad Utility for Windows](#)



## 6.4.19 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 [Figure 6.27](#).

Universal Command is used to execute UEMLoad on the remote UEM Server's system. The definition load file is redirected from standard input (`stdin`), which eliminates the step of copying the load file to the remote system before executing UEMLoad.

```
ucmd -cmd "uemload -add" -host rmthost -encryptedfile rmtacctinfo.enc  
<uemadd.txt
```

Figure 6.25 Redirect Definition File from stdin

### Components

[UEMLoad Utility for Windows](#)

[Universal Command Manager for Windows](#)

[Universal Event Monitor Server](#)

## 6.4.20 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 [Figure 6.27](#).

Universal Command is used to execute UEMLoad on the remote UEM Server's system. It redirects standard input (stdin) from a data set allocated to the UNVIN ddname. This eliminates the step of copying the data set to the remote system before executing UEMLoad.

```
//STEP1    EXEC UCMDPRC
//UNVIN    DD  DISP=SHR,DSN=MY.UEM.DATA(UEMDEF)
//SYSIN    DD  *
-host      dallas
-userid    joe
-pwd       ahzidaeh
-cmd       "uemload -add"
```

Figure 6.26 Redirect Definition File from STDIN (for z/OS)

### Components

[UEMLoad Utility for Windows](#)

[Universal Command Manager for z/OS](#)

[Universal Event Monitor Server](#)

## 6.4.21 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 Stonebranch Solutions 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 "optva11 ""optva12 optva12a"" optva13"**).

The **script** keyword can be used in lieu of a **begin\_script/end\_script** block, in which case the contents of the specified file will be written to the event handler.

A sample definition file for Windows is shown in [Figure 6.27](#).

```
# Indented lines are for illustration only. Leading spaces are
# ignored by UEMLoad. Defaults will be used for any omitted
# values.

# Start of parameters for an event definition with an ID of
# "win_event_sample".

begin_event
    event_id win_event_sample
    event_type FILE
    comp_name uems
```

```
state enable
inact_date_time 2004.12.31,23:59
triggered_id script_sample
filespec "uem*.dat"
rename_file yes
rename_filespec "$ (compname).$(compid).$(date).$(seqno)"
end_event

# End of parameters for event definition "win_event_sample".

# Start of parameters for an event handler with an ID of
# "win_script_sample".

begin_handler
  handler_id script_sample
  handler_type SCRIPT
  maxrc 0
  userid uemuser
  begin_script
    stmt "@echo off"
    stmt ""
    stmt "dir /-p/o/s ""C:\Program Files""
  end_script
  script_type bat
end_handler

# End of parameters for event handler "win_script_sample".

# Start of parameters for an event definition with an ID of
# "win_cmd_sample".

begin_handler
  handler_id cmd_sample
  maxrc 0
  userid uemuser
  cmd "C:\Documents and Settings\uemuser\TEST.BAT"
end_handler

# End of parameters for event definition "win_cmd_sample".
```

Figure 6.27 Definition File Sample - Windows

## Components

### UEMLoad Utility for Windows

## 6.4.22 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
```

Figure 6.28 Using a Stored Event Handler Record

### Components

[Universal Event Monitor Manager for UNIX](#)

[Universal Event Monitor Server for Windows](#)

## 6.4.23 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"
```

Figure 6.29 Handling an Event with a Script

Figure 6.30, below, illustrates the contents of the `/UEMScripts/h_001.txt` file.

```
:: Sample script h_001.txt
@echo off

:: Program variables
set parmCtr=1

:: Loop through parameter list

:: **** Start of loop ****
:BeginLoop
if ""%1""="" goto EndLoop

:DisplayParm
echo Parm %parmCtr%: %1

:: Shift the next parm
shift
set /a parmCtr+=1

:: Go back to the top
goto BeginLoop

:: **** End of loop ****
:EndLoop
```

Figure 6.30 Contents of Sample Script File

## Components

[Universal Event Monitor Manager for UNIX](#)

[Universal Event Monitor Server for Windows](#)

## 6.4.24 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 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 -a1R /uem files'` command.

Note that the `-expired` option also contains the `-options` parameter. In this example, the `-options` parameter redirects the output of the `'ls -a1R /uem files'` command to a file in uemuser's home directory called `uemtest.log`.

```
uem -host uemhost -event_type file -filespec uemtest.dat
-userid uemuser -pwd uemusers_password
-inact_date_time +1
-expired -cmd 'ls -a1R "/uem files"' -options '>uemtest.log 2>&1'
```

Figure 6.31 Handling an Expired Event

### Components

[Universal Event Monitor Manager for UNIX](#)

[Universal Event Monitor Server for UNIX](#)



## 6.4.25 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 `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
```

Figure 6.32 Adding a single event definition record.

### Components

[UEMLoad Utility for UNIX](#)

[Universal Event Monitor Server for UNIX](#)

## 6.4.26 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"
```

Figure 6.33 Adding a single event handler record.

### Components

[UEMLoad Utility for UNIX](#)

[Universal Event Monitor Server for UNIX](#)

[Universal Encrypt](#)

## 6.4.27 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 within quotes (that is: `\*` or `"*"`, respectively).

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

Figure 6.34 Listing all event definition records.

**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.

### Components

[UEMLoad Utility for UNIX](#)

## 6.4.28 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 [Figure 6.27](#).

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

Figure 6.35 Exporting all Event and Handler Records

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

### Components

[UEMLoad Utility for UNIX](#)

## 6.4.29 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
```

Figure 6.36 List a Single Event Handler Record

Figure 6.37, below, illustrates sample output for this command. (The values shown are those that could be expected if the record were added using the command shown in Figure 6.18.)

In this specific instance, the user ID contained in `encrypted.file` (from Figure 6.18) is `sparkie`, and the record was added by the user account with an ID of `sbuser`.

```
UNV3659I Connecting to local broker on port 7887.
UNV3406I Universal Event Monitor Server component 1117035117 started.
UNV3666I Load request started at 11:32:45 AM 05/25/2005.

Event Handler(s):
=====
Handler ID.....: dirlist
Max Acceptable Return Code.: 0
User ID.....: sparkie
Command.....: ls -al
Last Modified On.....: 05/25/2005 11:32:06 AM
Last Modified By.....: sbuser

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

Figure 6.37 Sample List Output

## Components

### [UEMLoad Utility for UNIX](#)

## 6.4.30 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??
```

Figure 6.38 Using Wildcards to List Records

### Components

[UEMLoad Utility for UNIX](#)

## 6.4.31 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 [Figure 6.27](#).

A definition file allows multiple records to be added to the event definition and/or event handler databases at the same time. When no definition file is used, only a single record can be added to the database(s).

```
uemload -add -deffile uemadd.txt
```

Figure 6.39 Add Database Record(s) Using a Definition File

### Components

[UEMLoad Utility for UNIX](#)

## 6.4.32 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 [Figure 6.27](#).

Universal Command is used to execute UEMLoad on the remote UEM Server's system. The definition load file is redirected from standard input (stdin), which eliminates the step of copying the load file to the remote system before executing UEMLoad.

```
ucmd -cmd "uemload -add" -host rmthost -encryptedfile rmtacctinfo.enc  
<uemadd.txt
```

Figure 6.40 Redirect Definition File from stdin

### Components

[Universal Command Manager for UNIX](#)

[UEMLoad Utility for UNIX](#)

[Universal Event Monitor Server for UNIX](#)



### 6.4.33 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 [Figure 6.27](#).

Universal Command is used to execute UEMLoad on the remote UEM Server's system. It redirects standard input (stdin) from a data set allocated to the UNVIN ddname. This eliminates the step of copying the data set to the remote system before executing UEMLoad.

```
//STEP1    EXEC UCMDPRC
//UNVIN    DD  DISP=SHR,DSN=MY.UEM.DATA(UEMDEF)
//SYSIN    DD  *
-host      dallas
-userid    joe
-pwd      ahzidaeh
-cmd      "/opt/universal/bin/uemload -add"
/*
```

Figure 6.41 Redirect Definition File from STDIN (for z/OS)

#### Components

[Universal Command Manager for z/OS](#)

[UEMLoad Utility for UNIX](#)

[Universal Event Monitor Server for UNIX](#)

## 6.4.34 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 Stonebranch Solutions 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 Windows is shown in [Figure 6.27](#).

```
# Indented lines are for illustration only. Leading spaces are
# ignored by UEMLoad. Defaults will be used for any omitted
# values.

# Start of parameters for an event definition with an ID of
# "unix_event_sample".

begin_event
    event_id unix_event_sample
    event_type FILE
    comp_name uems
```

```
state enable
inact_date_time 2004.12.31,23:59
triggered_id unix_script_sample
filespec 'uem*.dat'
rename_file yes
rename_filespec '$(compname).$(compid).$(date).$(seqno)'
```

end\_event

# End of parameters for event definition "unix\_event\_sample".

# Start of parameters for an event handler with an ID of  
# "unix\_script\_sample".

```
begin_handler
  handler_id unix_script_sample
  handler_type SCRIPT
  maxrc 0
  userid uemuser
  begin_script
    stmt "#!/bin/sh"
    stmt ""
    stmt 'ls -al "/home/uem user"'
  end_script
  script_type bat
end_handler
```

# End of parameters for event handler "unix\_script\_sample".

# Start of parameters for an event definition with an ID of  
# "unix\_cmd\_sample".

```
begin_handler
  handler_id unix_cmd_sample
  maxrc 0
  userid uemuser
  cmd '/home/uem user/someapp'
end_handler
```

# End of parameters for event definition "unix\_cmd\_sample".

Figure 6.42 Definition File Sample - UNIX

## Components

### UEMLoad Utility for UNIX

---

CHAPTER 7  
Security

---

## 7.1 Overview

This chapter provides information on the Security feature of Infitran:

- [Security of Infitran Components](#)
- [Encryption](#)
- [Encryption Examples](#)
- [Universal Access Control List](#)
- [Universal Access Control List Examples](#)
- [X.509 Certificates](#)

## 7.2 Security of Infitran Components

Each component of Infitran is designed to be a secure system.

As the level of security rises, so does the administrative complexity of the system. Infitran balances the two, minimizing administrative complexity without sacrificing security.

This section identifies the security features inherent in the design for each component.

- [Universal Broker](#)
- [Universal Data Mover Manager Security](#)
- [Universal Data Mover Server](#)
- [Universal Event Monitor Manager](#)
- [Universal Control Manager](#)
- [Universal Control Server](#)
- [Universal Event Log Dump](#)
- [Universal Spool List](#)
- [Universal Spool Remove](#)

## 7.2.1 Universal Broker

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

Universal Broker security concerns are:

1. Access to Universal Broker files and directories
2. Access to Universal Broker configuration options
3. Account with which Universal Broker executes
4. Privacy and integrity of transmitted network data

### File Permissions

At a minimum, only trusted user accounts should have write access to the Universal Broker installation data sets. This most likely means only the administrators should have write access. For maximum security, only trusted accounts should have read access to these data sets.

#### z/OS

Universal Broker requires update access to its database files, which exist as HFS- or zFS-allocated datasets mounted on the zOS Unix System Services (zOS USS) file system. The Broker accesses HFS-allocated datasets using the **UNVDB** and **UNVSPool** ddnames in its STC JCL. The Broker accesses zFS-allocated datasets via its **UNIX\_DB\_DATA\_SET** and **UNIX\_SPOOL\_DATA\_SET** configuration options.

#### Windows

Universal Broker requires write access to its primary install directory (that is, `.\Universal\UBroker`), which serves as its default trace file location.

The Broker requires full control over the database directory (that is, `.\Universal\spool`), along with all subdirectories and files under that location.

When the Broker executes as a console application with its message destination set to **logfile**, the Broker requires full control over the `.\Universal\Broker\log` directory and all `.log` files within it. The Universal Broker Windows service always writes its message to the Windows event log, which means that it requires no write access to a log directory or any other of its installation subdirectories and files.

#### UNIX

All files that the Broker creates or updates are located in the `/var/opt/universal` directory. This means that the Broker does not need write access to its installation directory or subdirectories.

Universal Broker requires full control (read, write, remove, and add) of the `/var/opt/universal` directory and its subdirectories. The Broker creates spool files, trace files, and log files in this directory. Users accounts other than the administrator accounts do not require access to this directory.

The Broker configuration options can be changed to use directories other than `/var/opt/universal`. If this is the case, the same permissions must be set up for these specified directories.

### IBM i

At a minimum, limit non-trusted user accounts to object authority of use to the Universal Broker product library, **UNVPRD420**; the product temporary library, **UNVTMP420**; the command reference library, **UNVCMDFREF**; the universal spool library, **UNVSPL420**; and all objects within these libraries.

For maximum security, only trusted accounts (administrators and the **UNVUBR420** profile) should have management, existence, alter, add, update, or delete authority to these objects. As a reminder, the system value **QCRTAUT** controls public access authority to created objects unless overridden by specific commands.

## Configuration Files

Only trusted user accounts should have write, create or delete access to the Broker configuration files or any of the directories in the configuration file directory search list.

### Windows

Although you can edit Stonebranch Solutions configuration files with any text editor (for example, Notepad), we recommend using the Universal Configuration Manager Control Panel application set configuration options.

The 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 (see [Section 8.4 Universal Configuration Manager](#)). It also directs the Broker to refresh its cache of Infitran component configuration settings, making it unnecessary to issue a separate configuration REFRESH request via the Universal Control utility.

## Universal Access Control List

Universal Broker uses the Universal Access Control List (UACL) as an extra layer of security. The UACL contains entries (that is, rules) that permit or deny access to the Universal Broker (see [Section 7.5 Universal Access Control List](#) for details).

Universal Broker reads the UACL entries when the program is started. If the UACL file is changed, the new entries can be activated either by:

- Stopping and starting Universal Broker.
- Sending Universal Broker a Universal Control configuration refresh request, which instructs Universal Broker to reread all of its configuration files, including the UACL file (see [Section 8.5 Configuration Refresh](#)).

### Windows

Although you may edit the UACL file with any text editor (for example, Notepad), we recommend that you maintain UACL entries using the Universal Configuration Manager Control Panel application (see [Section 8.4 Universal Configuration Manager](#)). The Universal Configuration Manager sends a configuration refresh request to the Universal Broker. Updated values take effect immediately, making it unnecessary to recycle the Broker to apply UACL changes.

Via this method, a configuration refresh request is sent to Universal Broker, and any new entries take effect immediately. There is no need to stop and restart the Broker in order for the changes to be applied.

## Universal Broker User Account

Each Infitran component that Universal Broker spawns inherits the Broker's account credentials. Occasionally, Infitran components must perform privileged operations, such as establishing a file transfer environment using a local user account's credentials.

On some platforms, this means that the Broker must execute with an account whose inherited credentials allow the spawned components to perform these operations.

On other platforms, the Broker may be execute with a lesser-privileged user, provided the components are configured in way that permits them to elevate their privileges when necessary.

The section contains platform-specific requirements to consider when setting the Broker's user account.

### z/OS

The Universal Broker started task may execute with any OMVS user ID provided that account has read access to the **BPX . DAEMON**, **BPX . SUPERUSER**, and **BPX . JOBNAME** resources in the FACILITY class.

The Broker user account is typically configured at install time. Complete details for configuring the Broker user account are in the Stonebranch Solutions Installation Guide.

### Windows

The Universal Broker Windows service must execute with the Local System account. Executing the Broker as Local System ensures that it and the components that it spawns have the privileges they require.

### UNIX

Although Universal Broker itself does not require super-user privileges, some Infitran server components (for example, UDM Server and UEM Server) may require super user authority to perform certain privileged operations.

Since the component inherits its user ID from Universal Broker, one of the following is required:

- Universal Broker must execute as **root**.
- **root** must own the Infitran Server application file (for example, **udmsrv** or **uemsrv**), and the Infitran Server application file must have its "set user ID on execution" bit (**setuid on exec**) set (for example, **chmod u+s udmsrv**).

If Universal Broker is started as a daemon at system start-up time, it is started with a user ID of **root**. Universal Broker and all its components then will have sufficient authority.



**IBM i**

Universal Broker for IBM i runs with the **UNVUBR420** user profile, which is created at product installation time. Any component started by Universal Broker inherits this user profile.

By default, the **UNVUBR420** user profile has \*ALLOBJ, \*JOBCTL, and \*SPLCTL authority. Unless the user profile is modified as described in the following section, \*ALLOBJ authority is required for a component to switch its user profiles based on the request it is servicing. \*JOBCTL authority is required for internal control and should not be removed. The **UNVUBR420** user profile requires \*SPLCTL authority to provide Universal Submit Job job logs in specific, limited situations. (See the [Stonebranch Solutions Utilities Reference Guide](#) for information on Universal Submit Job.)

Any other product or user should not use the **UNVUBR420** user profile. By default, users cannot access the system with the **UNVUBR420** profile.

**Removing \*ALLOBJ Authority from UNVUBR420 User Profile**

Given the extensive authority allowed by \*ALLOBJ special authority, it is desirable to avoid its use when possible. As of PTF 0UC0126 for V1R2M1, it is possible to remove \*ALLOBJ special authority from the **UNVUBR420** user profile. However, by removing \*ALLOBJ from the **UNVUBR420** user profile, the administrative complexity is increased.

The following describes the steps that are required to use Universal Command with \*ALLOBJ special authority removed from the **UNVUBR420** user profile.

1. If the following objects do not have \*USE Public Authority, the **UNVUBR420** user profile must be given \*USE authority:
  - QSYS/QSYGETPH
  - QSYS/QWTSETP
  - QSYS/QWCRJBST
  - QSYS/QUSRMBRD

This can be accomplished with the following command:

```
===> EDTOBJAUT OBJ(QSYS/object_name) OBJTYPE(*PGM)
```

From the resulting screen, use F6 to add user **UBROKER** and give it \*USE authority.

2. **UNVUBR420** user profile must be given \*USE authority to the user profile objects of all user profiles that will be using the universal command server on the IBM i.

This can be accomplished with the following command:

```
===> EDTOBJAUT OBJ(QSYS/user_profile_name) OBJTYPE(*USRPRF)
```

From the resulting screen, use F6 to add user **UBROKER** and give it \*USE authority.

3. Use the following command to remove the **UNVUBR420** user profile \*ALLOBJ authority:
 

```
===> CHGUSRPRF USRPRF(UNVUBR420) SPCAUT(*JOBCTL *SPLCTL)
```

**Removing \*SPLCTL Authority from UNVUBR420 User Profile**

Use the following command to remove the **UNVUBR420** user profile \*SPLCTL authority:

```
===> CHGUSRPRF USRPRF(UNVUBR420) SPCAUT(*JOBCTL *ALLOBJ)
```

**Removing \*ALLOBJ and \*SPLCTL Authorities from UNVUBR420 User Profile**

Use the following command to remove all special authority from the **UNVUBR420** user profile:

```
===> CHGUSRPRF USRPRF(UNVUBR420) SPCAUT(*JOBCTL)
```

(Please refer to the previous two sections for additional information.)

## 7.2.2 Universal Data Mover Manager Security

Universal Data Mover is designed to be a secure system. As the level of security rises, so does the administrative complexity of the system. Universal Data Mover has balanced the two to avoid the administrative complexity with a minimum sacrifice to security.

Universal Data Mover security concerns are:

1. Access to Universal Data Mover files and directories
2. Access to Universal Data Mover configuration files
3. Universal Data Mover user account
4. Privacy and integrity of transmitted network data
5. User authentication

### File Permissions

Only trusted user accounts should have permission to write to the Universal Data Mover installation directory and subdirectories, and all files within those directories.

#### z/OS

Data set permissions:

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

#### IBM i

Object Permissions:

Only administrator accounts should have write permission to the following Stonebranch Solutions libraries (and all objects within these libraries):

- Installation library, **UNVPRD420** (by default)
- Product temporary library, **UNVTMP420**
- Universal spool library, **UNVSPL420**

For maximum security, only trusted accounts (administrators and the **UNVUBR420** profile) should have management, existence, alter, add, update, and delete authority to these objects.

Note: System value **QCRTAUT** controls public access authority to created objects unless overridden by specific commands.

### Configuration Files

Only trusted user accounts should have write access to the Universal Data Mover Manager configuration files.

#### Windows

Although you may edit configuration files with any text editor (for example, Notepad), we recommend that you manage configuration options using the Universal Configuration Manager Control Panel application. Only user accounts in the Administrator group can execute the Universal Configuration Manager (see Section [8.4 Universal Configuration Manager](#)).

## 7.2.3 Universal Data Mover Server

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

Universal Data Mover Server security concerns are:

1. Access to product data sets
2. Access to Stonebranch Solutions configuration files
3. Universal Broker user account
4. Privacy and integrity of transmitted network data
5. User authentication

### File Permissions

Only trusted user accounts should have write permission to the Universal Data Mover Server installation directory and subdirectories, and all of the files within them.

#### z/OS

Only trusted user accounts should have write permission to the Universal Data Mover Server installation data sets. No general user access is required.

#### IBM i

Object Permissions

Only administrator accounts should have write permission to the following Stonebranch Solutions libraries (and all objects within these libraries):

- Installation library, **UNVPRD420** (by default)
- Product temporary library, **UNVTMP420**
- Universal spool library, **UNVSPL420**

For maximum security, only trusted accounts (administrators and the UNVUBR420 user profile) should have management, existence, alter, add, update or delete authority to these objects. As a reminder, the system value **QCRTAUT** controls public access authority to created objects unless overridden by specific commands.

### Configuration Files

Only trusted user accounts should have write access to the Universal Data Mover Server configuration files.

#### Windows

Although you may edit configuration files with any text editor (for example, Notepad), we recommend that you manage configuration options using the Universal Configuration Manager Control Panel application. Only user accounts in the Administrator group can execute the Universal Configuration Manager (see Section [8.4 Universal Configuration Manager](#)).

## Universal Data Mover Server User ID

Universal Data Mover Server requires read access to its installation directory and its working directory (defined in the component definition).

### z/OS

Universal Data Mover Server requires read access to its installation data sets and its HFS working directory (defined in the component definition).

### UNIX

If user security is activated, the Server requires root access to create processes that execute with another user's identity. The Server security identity is inherited from the Broker. If the Broker is running with a non-root user ID, then the Server program must have the set user ID on execution permission set and root as owner.

## Universal Data Mover Server User Profile

If user security is activated, the UDM Server for IBM i requires, by default, **\*ALLOBJ** authority to switch user profiles. This **\*ALLOBJ** authority requirement may be removed. The UDM Server initially inherits authority from the **UNVUBR420** user profile. Following the switch to the user profile, the UDM Server runs under the authority of the user initiating the data transfer.

The **UNVUBR420** user profile requires **\*SPLCTL** authority in order to provide Universal Submit Job with job logs in specific limited situations. The **\*SPLCTL** authority requirement can be removed. Removing **\*SPLCTL** from the **UNVUBR420** user profile may prevent the job log processing in limited situations.

(See [Universal Broker User Account](#) in Section 7.2.1 [Universal Broker](#) for information on removing the **\*ALLOBJ** and **\*SPLCTL** authorities.)

## User Authentication

User authentication is the process of verifying that a user is known and valid to the system. The process used by UDM Server requires the user to provide a user name / ID and a password. The UDM Server passes the name / ID and password to the operating system for verification; this is referred to as logging on the user.

### Windows

For Windows, user authentication is optional. However, if security is enabled, a user name / ID and password are required in order to verify the user's credentials. (With security enabled, you transfer files using a specific user's security context.)

## UNIX

For UNIX, user authentication is optional. However, if security is enabled, a user name / ID and password are required in order to verify the user's credentials. With security enabled, you transfer files using a specific user's security context.

Universal Data Mover can use three different types of user authentication methods:

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

## HP-UX 11.00 and later

By default, supplemental group memberships are recorded in the `/etc/group` file. However, if an `/etc/logingroup` file exists, it governs all supplemental group memberships and effectively overrides the entries in `/etc/group`.

Note: `/etc/logingroup` is not required to record supplemental group membership. If `/etc/logingroup` does not exist, `/etc/group` is sufficient to record the groups in which a user belongs.

If any Stonebranch Solutions component fails to access system resources that are secured based on supplemental group membership, make sure that the authenticated user has an entry in `/etc/logingroup`, if that file exists. Otherwise, the default entry in `/etc/group` should be sufficient.

For more information about `/etc/logingroup`, please see the HP-UX system documentation.

## IBM i

For IBM i, user authentication is optional. However, if security is enabled, a user name / ID and password are required in order to verify the user's credentials. With security enabled, you transfer files using a specific user's security context.

---

## 7.2.4 Universal Event Monitor Manager

---

### File Permissions

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

#### IBM i

Eligible users of UEM require read access to the national language support library **SUNVNLS**, the configuration file **UNVCONF**, and the load library **SUNVLOAD**.

#### Windows

Authorized users of UEM require read access to the message catalogs (\*.umc files), which reside in the `.\Universal\n1s` directory. If UEM Manager is installed on an NTFS partition, these file permissions are set automatically during the installation.

#### UNIX

Authorized users of UEM require read access to the message catalogs (\*.umc files) in the `n1s` subdirectory of the primary Stonebranch Solutions installation directory.

### Data Privacy

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

For more information on the steps taken to protect transferred data, see [Chapter 16 Network Data Transmission](#).

### RACF Protection

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

See the Stonebranch Solutions 4.2.0 Installation Guide for complete details on installing and administering UEM Manager RACF profiles.

## Configuration Files

Only trusted user accounts should have write access to the Universal Event Monitor Manager configuration files.

### Windows

Although you may edit configuration files with any text editor (for example, Notepad), we recommend that you manage configuration options using the Universal Configuration Manager Control Panel application. Only user accounts in the Administrator group can execute the Universal Configuration Manager (see [Section 8.4 Universal Configuration Manager](#)).

## 7.2.5 Universal Event Monitor Server

### Data Privacy

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

For more information on the steps taken to protected transferred data, see [Chapter 16 Network Data Transmission](#).

### File Permissions

Only trusted user accounts should have write access to the UEM Server installation directory and subdirectories, and the files within them. Authorized users of UEM require read access to the message catalogs (\* .umc files), which reside in the `./universal/n1s` directory.

#### Windows

If UEM Server is installed on an NTFS partition, these file permissions are automatically set during installation.

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

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

### Configuration Files

Only trusted user accounts should have write access to the Universal Event Monitor Server configuration files.

#### Windows

Although you may edit configuration files with any text editor (for example, Notepad), we recommend that you manage configuration options using the Universal Configuration Manager Control Panel application. Only user accounts in the Administrator group can execute the Universal Configuration Manager (see [Section 8.4 Universal Configuration Manager](#)).



## User Authentication

### Windows

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

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

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

For information on configuring UEM Server to use this logon method, see the [LOGON\\_METHOD](#) configuration option in the Universal Event Monitor 4.2.0 Reference Guide.

### UNIX

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

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

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

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

### HP-UX 11.00 and later

By default, supplemental group memberships are recorded in the `/etc/group` file. However, if an `/etc/logingroup` file exists, it governs all supplemental group memberships and effectively overrides the entries in `/etc/group`.

Note: `/etc/logingroup` is not required to record supplemental group membership. If `/etc/logingroup` does not exist, `/etc/group` is sufficient to record the groups in which a user belongs.

If any Stonebranch Solutions component fails to access system resources that are secured based on supplemental group membership, make sure that the authenticated user has an entry in `/etc/logingroup`, if that file exists. Otherwise, the default entry in `/etc/group` should be sufficient.

For more information about `/etc/logingroup`, please see the HP-UX system documentation.

## 7.2.6 Universal Control Manager

---

### File Permissions

Only trusted user accounts should have write permission to the Universal Control Manager installation directory and subdirectories, and all of the files within them. This most likely means that only the administrator group should have write access.

Eligible users of Universal Control require read access to the message catalogs (\*.umc files) in the NLS directory.

#### Windows

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

#### z/OS

Data set permissions: Eligible users of Universal Control require read access to:

- National language support library **SUNVNLS**
- Load library **SUNVLOAD**

### Configuration Files

Only trusted user accounts should have write access to the Universal Control Manager configuration files.

#### Windows

Although you may edit configuration files with any text editor (for example, Notepad), we recommend that you manage configuration options using the Universal Configuration Manager Control Panel application. Only user accounts in the Administrator group can execute the Universal Configuration Manager (see Section [8.4 Universal Configuration Manager](#)).

### Universal Configuration Manager

Universal Control Manager for Windows configuration options can be set by the Universal Configuration Manager. To protect access to the configuration settings, the Universal Configuration Manager can be executed only by accounts in the Administrator group.

For more information on the Universal Configuration Manager, see Section [8.4 Universal Configuration Manager](#).

## RACF Protection

The Universal Control Manager for z/OS verifies a user's access to a RACF general resource profile. The resource profile controls a user's access to execute a control request on a remote host.

See the Stonebranch Solutions 4.2.0 Installation Guide for complete details on installing and administering Universal Control Manager RACF profiles.

## 7.2.7 Universal Control Server

### File Permissions

Only trusted user accounts should have write permission to the Universal Control Server installation directory and subdirectories, and all of the files within them.

#### Windows

Eligible users of UCTL require read access to the message catalogs (\*.umc files) in the `n1s` subdirectory of the Stonebranch Solutions installation directory.

If security is activated, all eligible users of UCTL require permission to create directories in the UCTL Server working directory. A directory named after the user ID requesting the command is created for each user. The directory is created while impersonating the user; hence, it is created using the user's security account.

Home directories are created with permissions giving the user full control of both the directory and the files within them.

### Configuration Files

Only trusted user accounts should have write access to the Universal Control Server configuration files.

#### Windows

Although you may edit configuration files with any text editor (for example, Notepad), we recommend that you manage configuration options using the Universal Configuration Manager Control Panel application. Only user accounts in the Administrator group can execute the Universal Configuration Manager (see Section [8.4 Universal Configuration Manager](#)).

### Universal Control Server User ID

Universal Control Server requires read access to its installation directory and its working directory (defined in the component definition). The Universal Control Server security identity is inherited from the Universal Broker.

#### z/OS

UCTL Server requires read access to its installation data sets and its HFS working directory (defined in the component definition).

#### IBM i

The associated user profile (**UNVUBR420**) provides \*ALLOBJ authority.

## User Authentication

User authentication is the process of verifying that a user is a known and valid user. The process used by Universal Control Server requires the user to provide a user name / ID and a password. The Universal Control Server passes the name / ID and password to the operating system for verification; this is referred to as logging on the user.

### Windows

Windows provides two primary types of log on processes: batch and interactive.

A user must be given the right to log on as a batch job in order for the user to do a batch log on. All users can do an interactive log on. (See Section [6.7 LOGON\\_METHOD](#) in the Stonebranch Solutions Utilities 4.2.0 Reference Guide for more details.)

---

## 7.2.8 Universal Event Log Dump

---

No special security access is required to run Universal Event Log Dump (UELD). However, accessing the event logs and setting configuration options may require some special security considerations.

### Event Log Access

The system and application event logs may be read by all user accounts. The security log can only be accessed by accounts with Administrator privileges. Administrator privileges are also required to clear any of the event logs.

### Configuration Files

Only trusted user accounts should have write access to the Universal Event log Dump configuration files.

#### Windows

Although you may edit configuration files with any text editor (for example, Notepad), we recommend that you manage configuration options using the Universal Configuration Manager Control Panel application. Only user accounts in the Administrator group can execute the Universal Configuration Manager (see Section [8.4 Universal Configuration Manager](#)).

---

## 7.2.9 Universal Spool List

---

The account used to execute the Universal Spool List utility must have read access to the database files listed in Chapter [14 Databases](#).

---

## 7.2.10 Universal Spool Remove

---

The user account used to run the Universal Spool Remove utility must have read/write access to the database files.

## 7.3 Encryption

The **Universal Encrypt** utility lets you encrypt passwords for remote servers accessed by Infitran.

### 7.3.1 Encrypting Password Files

---

Passwords 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 to the platform or server from which they will be used. This means that applications development, platform administrators, and security administrators can encrypt passwords in their own environments.

Universal Encrypt will encrypt passwords with either:

- 56-bit DES
- 256-bit AES

Universal Encrypt reads unencrypted passwords from its standard input and writes the encrypted version to its standard output. Universal Encrypt actually encrypts the command options used by Infitran so the password needs to be prefixed with `-PWD` or `-pwd`.

Unencrypted files are text files and contain comments that can be edited if required. Lines within the Unencrypted 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.

---

## 7.3.2 Transferring Encrypted Password Files between Servers

---

Files encrypted via Universal Encrypt are text files. They can be transferred between servers, using FTP or similar tools, in text mode.

Email also can be used between like systems, such as Windows and 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 Infitran, such as [Universal Access Control List](#) and [X.509 Certificates](#). These can be further used to ensure that access to servers is properly controlled.



---

## 7.4 Encryption Examples

This section provides examples of how to use Universal Encrypt to encrypt password files. Each example will encrypt a case sensitive password (**P@ssW0rd**) using AES 256 encryption.

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

---

### z/OS

[Creating Encrypted Files for z/OS](#)

[Using Encrypted Password File on z/OS](#)

---

### Windows

[Creating Encrypted Files for Windows](#)

[Using Encrypted Password File on Windows](#)

---

### UNIX

[Creating Encrypted Files for UNIX](#)

[Using Encrypted Password File on UNIX](#)

---

### IBM i

[Creating Encrypted Files for IBM i](#)

[Using Encrypted Password File on IBM i](#)

## 7.4.1 Creating Encrypted Files for z/OS

Assume that a command file named **MY.CLEAR.CMDFILE** contains the following data:

```
-userid Thomas -pwd thames
```

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**.

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

```
# Universal Encrypt
# Date . . . . . : Thu Nov  3 07:29:03 2010
# User . . . . . : T02JAH1
# Host . . . . . : hosta.acme.com
# Program . . . . : uencrypt 4.2.0 Level 5 Release Build 130
# Encryption . . . : AES 256-bit

1F7DAF62583C813EA874CA168FF626C348F7BF171477D380D9A2FFFED33C539B71B4206EA5021F
D92CDFDD931C3B88B9CD711A4693EFE6B49FAE9431E9C946F7F35C9B4C31335BFB3F97F0686EFF
37068245A6B58CBFE2ADE32997A132C4114AC52CD615B2E7E8672ED0BF9867CA13B1
```

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

### Components

[Universal Command Manager for z/OS](#)

[Universal Encrypt](#)

## 7.4.2 Using Encrypted Password File on z/OS

For z/OS, the Universal Command Manager [COMMAND\\_FILE\\_ENCRYPTED](#) 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
```

Figure 7.1 z/OS -encryptedfile example

### Components

[Universal Command Manager for z/OS](#)

[Universal Encrypt](#)

## 7.4.3 Creating Encrypted Files for Windows

Assume that a command file named `cmdfile.clear` contains the following data:

```
-userid Thomas -pwd thames
```

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

```
uencrypt -key MYKEY123 -aes yes <cmdfile.clear >cmdfile.encrypt
```

The resulting encrypted command file is written to file `cmdfile.encrypt`.

The figure below illustrates the contents of `cmdfile.encrypt`.

```
# Universal Encrypt
# Date . . . . . : Thu Nov  3 07:29:03 2010
# User . . . . . : T02JAH1
# Host . . . . . : hosta.acme.com
# Program . . . . : uencrypt 4.2.0 Level 5 Release Build 130
# Encryption . . . : AES 256-bit

1F7DAF62583C813EA874CA168FF626C348F7BF171477D380D9A2FFFD33C539B71B4206EA502
1FD92CDFDD931C3B88B9CD711A4693EFE6B49FAE9431E9C946F7F35C9B4C31335BFB3F97F068
6EFF37068245A6B58CBFE2ADE32997A132C4114AC52CD615B2E7E8672ED0BF9867CA13B1
```

This encrypted command file now can be used by any Stonebranch Solutions command, on any operating system, by specifying the encryption key `MYKEY123`.

### Components

[Universal Command Manager for Windows](#)

[Universal Encrypt](#)

## 7.4.4 Using Encrypted Password File on Windows

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

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

Figure 7.2 Windows -encryptedfile example

### Components

[Universal Command Manager for Windows](#)

[Universal Encrypt](#)

## 7.4.5 Creating Encrypted Files for UNIX

Assume that a command file named `cmdfile.clear` contains the following data:

```
-userid Thomas -pwd thames
```

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

```
uencrypt -key MYKEY123 -aes yes <cmdfile.clear >cmdfile.encrypt
```

The resulting encrypted command file is written to file `cmdfile.encrypt`.

The figure below illustrates the contents of `cmdfile.encrypt`.

```
# Universal Encrypt
# Date . . . . . : Thu Nov  3 07:29:03 2010
# User . . . . . : T02JAH1
# Host . . . . . : hosta.acme.com
# Program . . . . : uencrypt 4.2.0 Level 5 Release Build 130
# Encryption . . . : AES 256-bit

1F7DAF62583C813EA874CA168FF626C348F7BF171477D380D9A2FFFD33C539B71B4206EA502
1FD92CDFDD931C3B88B9CD711A4693EFE6B49FAE9431E9C946F7F35C9B4C31335BFB3F97F068
6EFF37068245A6B58CBFE2ADE32997A132C4114AC52CD615B2E7E8672ED0BF9867CA13B1
```

This encrypted command file now can be used by any Stonebranch Solutions command, on any operating system, by specifying the encryption key `MYKEY123`.

### Components

[Universal Command Manager for UNIX](#)

[Universal Encrypt](#)

## 7.4.6 Using Encrypted Password File on UNIX

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

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

Figure 7.3 UNIX -encryptedfile example

### Components

[Universal Command Manager for UNIX](#)

[Universal Encrypt](#)

## 7.4.7 Creating Encrypted Files for IBM i

Assume that a command file named **MYLIB/QTXTSRC (TESTLOGIN)** contains the following data:

```
-userid Thomas -pwd tz74gan
```

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

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

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

The figure below illustrates the contents of **MYLIB/ENCRYPTEDF (ENCRYPTEDF)**.

```
# Universal Encrypt  
# Created on wed Feb 22 18:43:51 2010  
# Created by uencrypt 4.2.0 Level 0  
  
9ACB96416816600CB9D24C9072D80C11768B93CB0E79B944EC37D3495097AD793F97399220C9BB  
472DF1E04F5BA8909BCA6C8C72DFD3B706487B1713E6F73F5A0539F17076DEF6D14083EF6E7023  
158526E70BE3AF688579805DCAC0CFF1EB6A
```

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

### Components

[Universal Command Manager for IBM i](#)

[Universal Encrypt](#)



## 7.4.8 Using Encrypted Password File on IBM i

For the IBM i, the Universal Command Manager `COMMAND_FILE_ENCRYPTED` option (`ECMFILE` and `ECMMBR`) specify the location of the Uencrypted file.

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

Figure 7.4 IBM i -encryptedfile example

### Components

[Universal Command Manager for IBM i](#)

[Universal Encrypt](#)

## 7.5 Universal Access Control List

Many Infitran components utilize the Universal Access Control List (UACL) feature as an extra layer of security to the services they offer. The UACL determines if a request is denied or allowed to continue and can assign security attributes to the request.

The following Infitran components use the UACL feature:

- Universal Broker uses UACLs to permit or deny TCP/IP connections based on the remote host IP address (see the [Universal Broker Reference Guide](#) for complete details).
- Universal Data Mover Server uses UACLs to permit or deny Manager access based on the Managers IP address and user ID (see the [Universal Data Mover Reference Guide](#) for complete details).
- Universal Control Server uses UACLs to permit or deny Manager access based on the Managers IP address and user ID, and to control whether or not the Manager request requires user authentication (see Chapter 4 [Universal Control](#) in the [Stonebranch Solutions Utilities Reference Guide](#) for complete details).

This section of the Infitran User Guide describes the UACL capabilities in general, non-component specific terms.

## 7.5.1 UACL Configuration

---

The method used to configure UACL rules is platform dependent. The following sections discuss each of the methods.

### z/OS

All UACL rules are defined in library **UNVCONF**, member **ACLCFG00**. The Universal Broker allocates the UACL configuration data set to ddname **UNVACL**.

The UACL file syntax is the same as all other Stonebranch Solutions z/OS configuration files. See [Configuration File Syntax](#) for details.

### UNIX

All UACL rules are defined in one file, **uac1.conf**. This file is required for products utilizing UACL rules; otherwise, the product will not start. The configuration file consists of zero or more UACL entries.

The UACL file syntax is the same as all other Universal UNIX configuration files. See [Configuration File Syntax](#) for details.

### Windows

All UACL rules are stored in the configuration file.

UACL entries for each component are maintained using the Universal Configuration Manager (see Section [8.4 Universal Configuration Manager](#)).

### IBM i

All UACL rules are defined in file **unvconf** and member **uac1**. This file is required for products utilizing UACL rules, else the product will not start. The configuration file consists of zero or more UACL entries.

The UACL file is searched for in the same manner as all other product configuration files. See Section [8.2.4 Configuration File](#) for information on how configuration files are located.

The UACL file syntax is the same as all other Stonebranch Solutions for IBM i configuration files. See [Configuration File Syntax](#) for details.

---

## 7.5.2 UACL Entries

---

UACL entries are composed of two parts: type and rule.

- Type identifies the Stonebranch Solutions component for which the rule applies. For example, the Universal Broker product utilizes UACL rules of type `ubroker_access`.
- Rule defines the client's identity and the client's request for which the entry pertains and the security attributes it enforces.

UACL configuration file syntax is the same as all other configuration files, where the configuration file keyword corresponds to the UACL type part and the configuration file value corresponds to the UACL rule part.

The entire rule part of the UACL entry must be enclosed in quotation characters, not just a sub-field of the rule, if a space or tab is part of the value.

The correct syntax would be as follows:

```
udm_access "prod.host.name,MVS USER,user ,cmd,DSPLIB
QGPL,allow,auth"
```

For each client that connects and sends a request, Broker and Server components search UACL entries to find the best match for the client identity and the client request. Entries are searched in the order they are listed. The first entry found stops the search.

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

### Client Identification

Rule matching is based on the client identity and the client request.

There are two client identification methods:

1. X.509 certificate authentication.
2. Client IP address and reported user account.

### X.509 Certificate Authentication

X.509 certificates identify an entity. An entity can be a program, person, or host computer. When an X.509 certificate is authenticated, it authenticates that the entity is who it claims to be.

X.509 certificates are utilized in UACL entries by first mapping a client certificate to a UACL certificate identifier. The certificate identifier then is used in the UACL entries. A certificate identifier provides for:

1. Concise representation of certificates in UACL entries. There are a large number of certificate fields that may be used and many of the fields have lengthy, tedious naming formats. A certificate map only needs to be defined once and then the concise certificate identifier can be used in the UACL entries.
2. Mapping of one or more certificates to a single certificate identity. A group of entities that share a common security access level may be represented by one certificate identity reducing the number of UACL entries to maintain.

UACL certificate map entries are searched sequentially (that is, top to bottom) matching the client certificate to each entry until a match is found. The certificate map defines a set of X.509 certificate fields that may be used as matching criteria.

Table 7.1, below, defines the certificate map matching criteria.

Criteria	Description
SUBJECT	<p>Matches the X.509 <code>subject</code> field. The <code>subject</code> field is formatted as an X.501 Distinguished Name (DN). A DN is a hierarchical list of attributes referred to as Relative Distinguished Names (RDNs).</p> <p>RDNs are separated with a comma (,) by default. If a different separator is required (perhaps one of the RDN values uses a comma), start the DN with the different separator character. Valid separators are slash (/), comma (,) and period (.).</p> <p>Many RDN values can be used in a DN. Some of the most common values are:</p> <ul style="list-style-type: none"> <li>• C Country name</li> <li>• CN Common name</li> <li>• L Locality</li> <li>• O Organization</li> <li>• OU Organizational Unit</li> <li>• ST State</li> </ul> <p>The RDN attributes must be listed in the same order as they are defined in the certificate to be considered matched.</p> <p>A partial DN can be specified. All certificates that have a <code>subject</code> name that matches up to the last RDN are considered a match. This permits a group of certificates to be matched.</p> <p>The RDN attribute values can include pattern matching characters. An asterisk (*) matches 0 or more characters and a question mark (?) matches one character.</p> <p>Some example of SUBJECT values are:</p> <ul style="list-style-type: none"> <li>• <code>subject="C=US,ST=Georgia,O=Acme,CN=Road Runner"</code></li> <li>• <code>subject="C=US,ST=Georgia,O=Acme,CN=Road *"</code></li> <li>• <code>subject="C=US,ST=Georgia,O=Acme,CN=Road ?unner"</code></li> </ul> <p>Whether an RDN value is case sensitive or not depends on the format in which the value is stored. The certificate creator has some control over which format is used. All formats except for <code>printableString</code> are case sensitive.</p>

Criteria	Description
EMAIL	<p>Matches the X.509 <b>emailAddress</b> attribute of the <b>subject</b> field and <b>rfc822Name</b> of the <b>subjectAltName</b> extension value. Both fields format the email address as an RFC 822 <b>addr-spec</b> in the form of <b>identifier@domain</b>.</p> <p>The attribute values may include pattern matching characters. An asterisk (*) matches 0 or more characters and a question mark (?) matches one character.</p> <p>Some example EMAIL values are:</p> <ul style="list-style-type: none"> <li>• <b>email=user1@acme.com</b></li> <li>• <b>email=*@acme.com</b></li> <li>• <b>email=user?@acme.com</b></li> </ul> <p>RFC 822 names are not case sensitive.</p>
HOSTNAME	<p>Matches the following X.509 fields in the order listed:</p> <ol style="list-style-type: none"> <li>1. <b>dNSName</b> of the <b>subjectAltName</b> extension value.</li> <li>2. <b>commonName (CN)</b> RDN attribute of the <b>subject</b> field's DN value.</li> </ol> <p>Some example HOSTNAME values are:</p> <ul style="list-style-type: none"> <li>• <b>hostname=bigfish.acme.com</b></li> <li>• <b>hostname=*.acme.com</b></li> </ul> <p>The values are not case sensitive.</p>
IP ADDRESS	<p>Matches the X.509 <b>iPAddress</b> field of the <b>subjectAltName</b> extension value.</p> <p>An example IPADDRESS value is:</p> <ul style="list-style-type: none"> <li>• <b>ipaddress=10.20.30.40</b></li> </ul>
SERIAL NUMBER	<p>Matches the X.509 <b>serialNumber</b> value.</p> <p>The value can be specified in a hexadecimal format by prefixing the value with <b>0x</b> or <b>0X</b>, otherwise, the value is considered a decimal format. For example, the value <b>0x016A392E7F</b> would be considered a hexadecimal format.</p> <p>An example SERIALNUMBER value is:</p> <ul style="list-style-type: none"> <li>• <b>serialnumber=0x7a2d52cbae</b></li> </ul>

Table 7.1 Certificate Map Matching Criteria

If a certificate map rule is found that matches the client certificate, the rule's identifier is assigned to the client's request. The certificate identifier is then used in matching certificate-based UACL entries.

Table 7.2, below, defines the certificate identifier field as used in UACL entries.

Criteria	Description
CERTID	<p>Matches the certificate identifier defined by the certificate map entry. The CERTID value has the following syntax:</p> <ul style="list-style-type: none"> <li>• An asterisk (*) matches 0 or more characters and a question mark (?) matches one character. For example, <b>AB*M</b> matches <b>ABCDM</b> and <b>ABM</b>. <b>AB?M</b> matches <b>ABCM</b>, but not <b>ABCDM</b>.</li> <li>• The comparison is case insensitive.</li> <li>• Pattern matching characters, such as the asterisk and question mark, are included in the text to be matched by prefixing them with a forward slash (/) character. For example, <b>A/*B</b> matches <b>A*B</b>. <b>A//B</b> matches <b>A/B</b>.</li> </ul>

Table 7.2 Certificate Identifier Field

### Client IP Address Identification

TCP/IP provides a method to obtain a client's IP address. The IP address typically identifies the host computer on which the client is executing. There are exceptions to this though. Networks can be configured with Network Address Translation (NAT) systems between the client and the Broker that hides the client's IP address. In addition to the client IP address, Stonebranch Solutions clients provide a user account name with which they are executing that is used to further refine the client's identity.

UACL entries are searched matching the client's IP address and user account to each entry until a match is found.

Table 7.3, below, defined possible matching criteria for IP address and user account client identification.

Criteria	Description
HOST	<p>Matches the TCP/IP address of the remote user.</p> <p>The HOST value has the following syntax:</p> <ul style="list-style-type: none"> <li>• Dotted numeric form of an IP address. For example, <b>10.20.30.40</b>.</li> <li>• Dotted numeric prefix of the IP addresses. For example, <b>10.20.30.</b> matches all IP addresses starting with <b>10.20.30</b>. The last dot (.) is required.</li> <li>• A <i>net/mask</i> expression. For example, <b>131.155.72.0/255.255.254.0</b> matches IP address range <b>131.155.72.0</b> through <b>131.155.73.255</b>. The <i>mask</i> and the host value are AND'ed together. The result must match <i>net</i>.</li> </ul> <p>Note: Contact your network administrator for calculation of the correct <i>net / mask</i> expression.</p> <ul style="list-style-type: none"> <li>• Host name for an IP address. For example, <b>sysa.abc.com</b>.</li> <li>• Host name suffix for a range of IP addresses. For example, <b>.abc.com</b> matches all host names ending with <b>abc.com</b>, such as, <b>sysa.abc.com</b>. The first dot (.) is required.</li> <li>• A value of <b>ALL</b> matches all IP addresses. The value must be uppercase.</li> </ul>
REMOTE_USER	<p>Matches the user name with which the remote user is executing as on the remote system.</p> <p>The REMOTE_USER value has the following syntax:</p> <ul style="list-style-type: none"> <li>• An asterisk (*) matches 0 or more characters and a question mark (?) matches one character. For example, <b>AB*M</b> matches <b>ABCDM</b> and <b>ABM</b>. <b>AB?M</b> matches <b>ABCM</b> but not <b>ABCDM</b>.</li> <li>• Control code <i>/c</i> switches off case-sensitivity and <i>/C</i> switches on case-sensitivity matching. The default is on. For example, <b>/cABC</b> matches <b>abc</b>. <b>/ca/Cbc</b> matches <b>Abc</b> but not <b>ABC</b>.</li> <li>• Pattern matching characters, such as the asterisk and question mark, are included in the text to be matched by prefixing them with a forward slash (/) character. For example, <b>A/*B</b> matches <b>A*B</b>. <b>A//B</b> matches <b>A/B</b>.</li> </ul>

Table 7.3 Client IP Address - Matching Criteria

## Request Identification

In addition to the client identity being used to search for UACL entries, the client's request may be part of the matching criteria. The exact request fields used is dependent on the component's UACL entry type.

[Table 7.4](#), below, lists a complete set of the request fields that are possible. See each component's UACL entry definitions for further details.

Criteria	Description
LOCAL_USER	<p>Matches the local user name with which the remote user is requesting to execute as on the local host. LOCAL_USER value has the following syntax:</p> <ul style="list-style-type: none"> <li>• An asterisk (*) matches 0 or more characters and a question mark (?) matches one character. For example, <b>AB*M</b> matches <b>ABCDM</b> and <b>ABM</b>. <b>AB?M</b> matches <b>ABCM</b> but not <b>ABCDM</b>.</li> <li>• Control code /c switches off case-sensitivity and /C switches on case-sensitivity matching. The default is on. For example, /c<b>ABC</b> matches <b>abc</b>. /ca/C<b>bc</b> matches <b>Abc</b> but not <b>ABC</b>.</li> <li>• Pattern matching characters, such as the asterisk and question mark, are included in the text to be matched by prefixing them with a forward slash (/) character. For example, A/<b>*B</b> matches <b>A*B</b>. A/<b>/B</b> matches <b>A/B</b>.</li> <li>• Variable name \$RMTUSER can be included in the value. The variable name itself is not case sensitive. \$RMTUSER and \$rmtuser are the same. The \$RMTUSER variable value is the user name with which the remote user is executing. It is the same value used in matching the REMOTE_USER field.</li> </ul> <p>A space character delimits the variable name, or it can be enclosed in parentheses (for example, \$(RMTUSER)), in which case it is delimited by the right parenthesis. This is useful if it is immediately followed by text.</p> <p>For example, if the remote user name is <b>TOM</b>, a LOCAL_USER value of \$RMTUSER will match if the local user name requested is also <b>TOM</b>. A LOCAL_USER value of \$(RMTUSER)01 will match if the local user name requested is <b>TOM01</b>.</p> <div style="background-color: #f4a460; padding: 2px; margin-top: 10px;"><b>Windows</b></div> <p>The LOCAL_USER value is not case sensitive since Windows user account names are not.</p>
REQUEST_TYPE	<p>Matches the type of request a Universal Command Manager is requesting. The REQUEST_TYPE value has the following syntax:</p> <ul style="list-style-type: none"> <li>• An asterisk (*) matches 0 or more characters and a question mark (?) matches one character. For example, <b>AB*M</b> matches <b>ABCDM</b> and <b>ABM</b>. <b>AB?M</b> matches <b>ABCM</b> but not <b>ABCDM</b>.</li> <li>• The comparison is case insensitive.</li> <li>• Pattern matching characters, such as the asterisk and question mark, are included in the text to be matched by prefixing them with a forward slash (/) character. For example, A/<b>*B</b> matches <b>A*B</b>. A/<b>/B</b> matches <b>A/B</b>.</li> </ul>



Criteria	Description
REQUEST_NAME	<p>The REQUEST_NAME field matches the name of a Universal Command Manager request. The REQUEST_NAME value has the following syntax:</p> <ul style="list-style-type: none"> <li>• An asterisk ( * ) matches 0 or more characters and a question mark ( ? ) matches one character. For example, <b>AB*M</b> matches <b>ABCDM</b> and <b>ABM</b>. <b>AB?M</b> matches <b>ABCM</b> but not <b>ABCDM</b>.</li> <li>• Case sensitivity depends on the REQUEST_TYPE and the operating system on which the Universal Command Server is executing. See the Server's Security section for the operating system in question.</li> <li>• Control code <b>/c</b> switches off case-sensitivity and <b>/C</b> switches on case-sensitivity matching. The default is on. For example, <b>/cABC</b> matches <b>abc</b>. <b>/ca/Cbc</b> matches <b>Abc</b> but not <b>ABC</b>.</li> <li>• Control code <b>/s</b> normalizes spaces and <b>/S</b> does not normalize spaces. Space normalization removes preceding and trailing spaces as well as reduce consecutive multiple spaces to a single space. The default is no space normalization. For example, <b>/sa b c</b> matches <b>a b c</b>. <b>/Sa b c</b> matches <b>a b c</b> but not <b>a bc</b>.</li> <li>• Pattern matching characters, such as the asterisk and question mark, are included in the text to be matched by prefixing them with a forward slash ( / ) character. For example, <b>A/*B</b> matches <b>A*B</b>. <b>A//B</b> matches <b>A/B</b>.</li> </ul>

Table 7.4 Request Fields

## Certificate-Based and Non Certificate-Based UACL Entries

Stonebranch Solutions components that support X.509 certificates define their UACL entries in two varieties:

1. Certificate-based entries
2. Non certificate-based entries

The two entry types are distinguished by their name. For example, **cmd\_cert\_access** is the certificate-based form of the entry and **ucmd\_access** is a non certificate-based entry. All entries follow the same format.

Certificate-based UACL entries are searched under the following conditions:

- Client provides an X.509 certificate that matches a certificate map entry.

Non certificate-based UACL entries are searched under the following conditions:

- Client provides an X.509 certificate and no certificate map entry matches.
- Client does not provide an X.509 certificate.

Either the certificate-based UACL entries or the non certificate-based UACL entries are searched, but not both.

---

## 7.6 Universal Access Control List Examples

This section provides the following UACL examples.

### z/OS

---

[Universal Broker for z/OS](#)

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

[Universal Control Server for z/OS](#)

### Windows

---

[Universal Broker for Windows](#)

[Universal Data Mover Server for Windows](#)

[Universal Control Server for Windows](#)

### UNIX

---

[Universal Broker for UNIX](#)

[Universal Data Mover Server for UNIX](#)

[Universal Control Server for UNIX](#)

### IBM i

---

[Universal Broker for IBM i](#)

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

[Universal Control Server for IBM i](#)

## 7.6.1 Universal Broker for z/OS

The following set of rules authorize the Universal Enterprise Controller at address 10.20.30, with update access to the product configuration files and setting of the configuration managed mode of the Broker, and denies all other connections.

```
remote_config_access    10.20.30.,allow,allow
remote-config_access    ALL,deny,deny
```

The following set of rules permit connections for the subnet 10.20.30 and denies all other connections.

```
ubroker_access         10.20.30.,allow
ubroker_access         ALL,deny
```

The following set of rules permit connections from host 10.20.30.40 and 10.20.30.50 and denies all other connections.

```
ubroker_access         10.20.30.40,allow
ubroker_access         10.20.30.50,allow
ubroker_access         ALL,deny
```

The following set of rules map X.509 certificates to certificate identifiers.

```
cert_map               id=joe,subject="/C=US/ST=Georgia/O=Acme, Inc./
                       OU=Sales/CN=Joe Black"
```

### Components

#### Universal Broker for z/OS

## 7.6.2 Universal Data Mover Server for z/OS

The following set of rules permit services for the subnet 10.20.30 and denies all other connections.

```
udm_access 10.20.30.,*,*,allow,auth
udm_access ALL,*,*,deny,auth
```

The following set of rules effectively permit connections from any host, but has limited access from host 10.20.30.40 to user TS1004 on that host. No host can execute commands as local user root. User TS1004 on host 10.20.30.40 can execute commands as local user tsup1004 without providing the password. Users TS1004 from host 10.20.30.40 can execute commands as any local user by providing the local user password.

```
udm_access 10.20.30.40,TS1004,tsup1004,allow,noauth
udm_access 10.20.30.40,TS1004,*,allow,auth
udm_access 10.20.30.40,*,*,deny,auth
udm_access ALL,*,root,deny,auth
```

### Components

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

## 7.6.3 Universal Control Server for z/OS

The following set of rules permit services for the subnet 10.20.30 and denies all other connections unless an X.509 certificate is presented that maps to certificate ID operations.

```

uctl_access      10.20.30.,*,*,allow,auth
uctl_access      ALL,*,*,deny,auth

uctl_cert_access operations,*,allow,auth
uctl_cert_access *,*,deny,auth

```

When no certificate is presented that maps to a certificate ID, the following set of rules effectively permit connections from any host, but has limited access from host 10.20.30.40 to user TS1004 on that host. No host can execute commands as local user root. User TS1004 on host 10.20.30.40 can execute commands as local user tsup1004 without providing the password. Users TS1004 from host 10.20.30.40 can execute commands as any local user by providing the local user password.

When a certificate is presented that maps to a certificate ID, certificate ID joe can request local user id TSUP1004 without a password. Certificate ID joe is allowed to execute commands with any other local user ID with a password. Certificate ID operations cannot run anything. All other certificate IDs can execute commands with any user ID except for SUPERID with a password.

```

uctl_access      10.20.30.40,TS1004,tsup1004,allow,noauth
uctl_access      10.20.30.40,TS1004,*,allow,auth
uctl_access      10.20.30.40,*,*,deny,auth
uctl_access      ALL,*,root,deny,auth

uctl_cert_access joe,tsup1004,allow,noauth
uctl_cert_access joe,*,allow,auth
uctl_cert_access operations,*,deny,auth
uctl_cert_access *,root,deny,auth

```

## Components

### Universal Control

## 7.6.4 Universal Broker for Windows

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

Figure 7.5, below, illustrates an example.

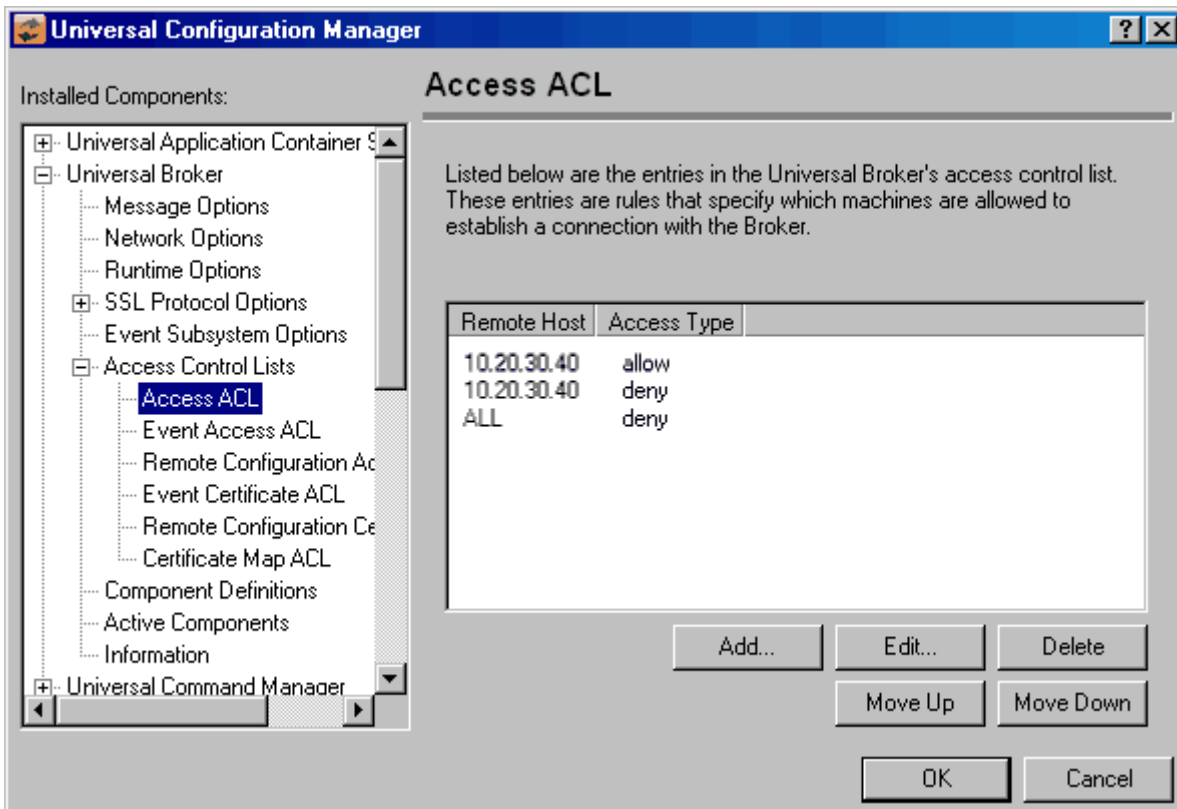


Figure 7.5 Universal Configuration Manager - Universal Broker - Access ACL

## Components

### Universal Broker for Windows

## 7.6.5 Universal Data Mover Server for Windows

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

Figure 7.6, below, illustrates an example. The set of ACL entries only allows connections from host 10.20.30.40 if the user on that host is TS1004. All other remote users will be blocked. TS1004 may run processes on the local system using any user account, provided the correct password is supplied. No processes may be run with Universal Data Mover using the Administrator account on the local system, regardless of where the request originated.

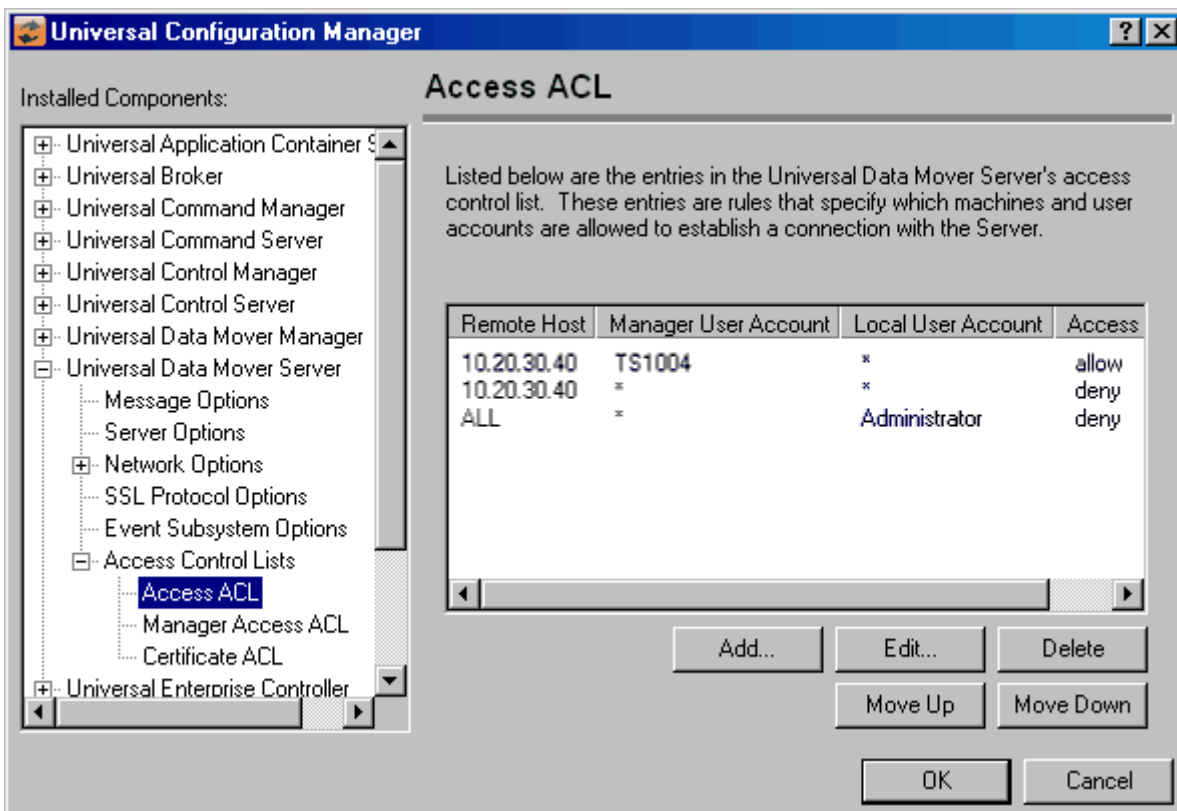


Figure 7.6 Universal Configuration Manager - Universal Data Mover Server - Access ACL

### Components

#### Universal Data Mover

## 7.6.6 Universal Control Server for Windows

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

Figure 7.7, below, illustrates an example. The set of ACL entries only allows connections from host 10.20.30.40 if the user on that host is TS1004. All other remote users will be blocked. TS1004 may run processes on the local system using any user account, provided the correct password is supplied. No processes may be run with Universal Command using the Administrator account on the local system, regardless of where the request originated.

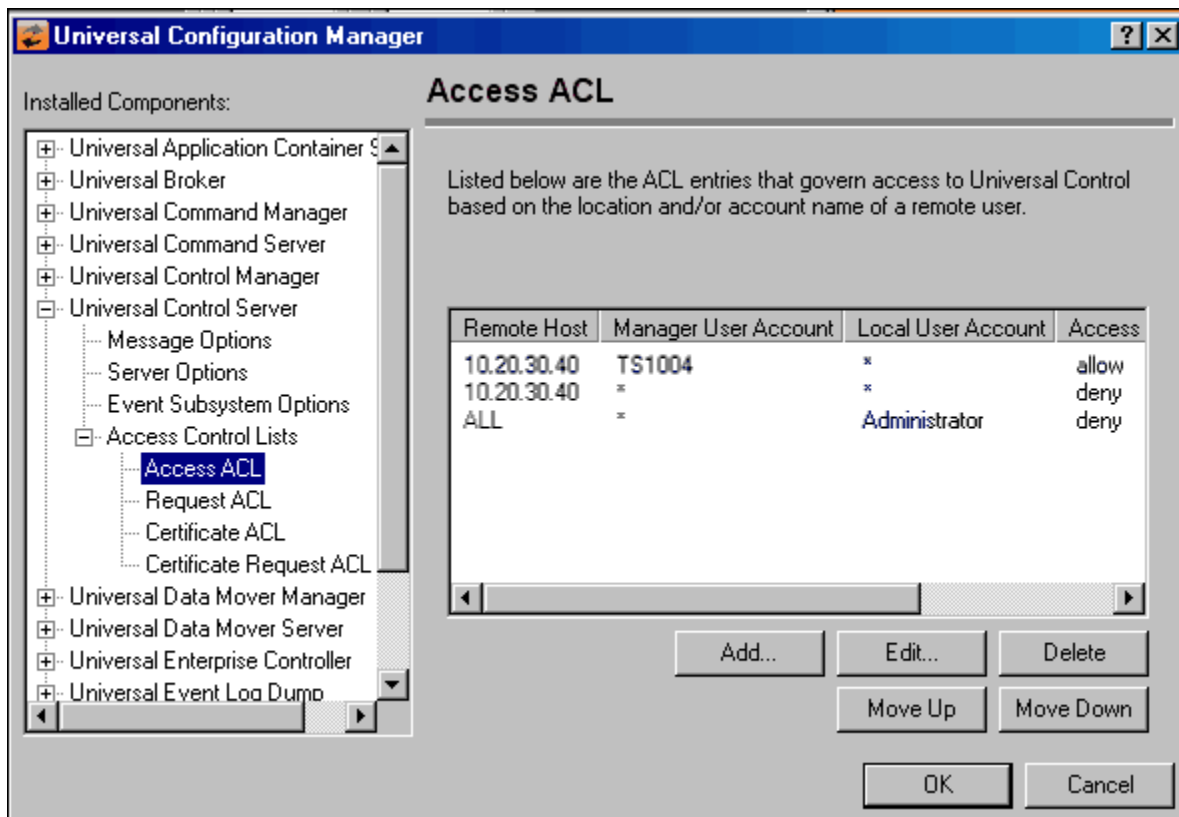


Figure 7.7 Universal Configuration Manager - Universal Control Server - Access ACL

### Components

#### Universal Control



## 7.6.7 Universal Broker for UNIX

The following set of rules is required to allow I-Management Console to access Universal Broker.

```
remote_config_access    10.20.30.,allow
remote-config_access    ALL,deny
```

The following set of rules permit connections for the subnet 10.20.30 and denies all other connections.

```
ubroker_access         10.20.30.,allow
ubroker_access         ALL,deny
```

The following set of rules permit connections from host 10.20.30.40 and 10.20.30.50 and denies all other connections.

```
ubroker_access         10.20.30.40,allow
ubroker_access         10.20.30.50,allow
ubroker_access         ALL,deny
```

The following set of rules map X.509 certificates to certificate identifiers.

```
cert_map                id=joe,subject="/C=US/ST=Georgia/O=Acme, Inc./"
```

### Components

#### Universal Broker for UNIX

## 7.6.8 Universal Data Mover Server for UNIX

The following set of rules permit services for the subnet 10.20.30 and denies all other connections.

```
udm_access    10.20.30.,*,*,allow,auth
udm_access    ALL,*,*,deny,auth
```

The following set of rules effectively permit connections from any host, but has limited access from host 10.20.30.40 to user TS1004 on that host. No host can execute commands as local user root. User TS1004 on host 10.20.30.40 can execute commands as local user tsup1004 without providing the password. Users TS1004 from host 10.20.30.40 can execute commands as any local user by providing the local user password.

```
udm_access    10.20.30.40,TS1004,tsup1004,allow,noauth
udm_access    10.20.30.40,TS1004,*,allow,auth
udm_access    10.20.30.40,*,*,deny,auth
udm_access    ALL,*,root,deny,auth
```

### Components

#### [Universal Data Mover Server for UNIX](#)

## 7.6.9 Universal Control Server for UNIX

The following set of rules permit services for the subnet 10.20.30 and denies all other connections unless an X.509 certificate is presented that maps to certificate ID operations.

```
uctl_access    10.20.30.,*,*,allow,auth
uctl_access    ALL,*,*,deny,auth

uctl_cert_access  operations,*,allow,auth
uctl_cert_access *,*,deny,auth
```

When no certificate is presented that maps to a certificate ID, the following set of rules effectively permits connections from any host, but has limited access from host 10.20.30.40 to user TS1004 on that host. No host can execute commands as local user root.

User TS1004 on host 10.20.30.40 can execute commands as local user tsup1004 without providing the password.

User TS1004 from host 10.20.30.40 can execute commands as any local user by providing the local user password.

When a certificate is presented that maps to a certificate ID, certificate ID joe can request local user id tsup1004 without a password. Certificate ID joe is allowed to execute commands with any other local user ID with a password. Certificate ID operations cannot run anything. All other certificate IDs can execute commands with any user ID except for root with a password.

```
uctl_access    10.20.30.40,TS1004,tsup1004,allow,noauth
uctl_access    10.20.30.40,TS1004,*,allow,auth
uctl_access    10.20.30.40,*,*,deny,auth
uctl_access    ALL,*,root,deny,auth

uctl_cert_access  joe,tsup1004,allow,noauth
uctl_cert_access  joe,*,allow,auth
uctl_cert_access  operations,*,deny,auth
uctl_cert_access  *,root,deny,auth
```

## Components

### Universal Control

## 7.6.10 Universal Broker for IBM i

The following set of rules permit connections for the subnet 10.20.30 and denies all other connections.

```
ubroker_access    10.20.30.,allow
ubroker_access    ALL,deny
```

The following set of rules permit connections from host 10.20.30.40 and 10.20.30.50 and denies all other connections.

```
ubroker_access    10.20.30.40,allow
ubroker_access    10.20.30.50,allow
ubroker_access    ALL,deny
```

The following set of rules map X.509 certificates to certificate identifiers.

```
cert_map          id=joe,subject="/C=US/ST=Georgia/O=Acme, Inc./
                  OU=Sales/CN=Joe Black"
```

### Components

#### Universal Broker for IBM i

## 7.6.11 Universal Data Mover Server for IBM i

The following set of rules permit services for the subnet 10.20.30 and denies all other connections.

```
udm_access    10.20.30.,*,*,allow,auth
udm_access    ALL,*,*,deny,auth
```

The following set of rules effectively permit connections from any host, but has limited access from host 10.20.30.40 to user TS1004 on that host. No host can execute commands as local user root. User TS1004 on host 10.20.30.40 can execute commands as local user tsup1004 without providing the password. Users TS1004 from host 10.20.30.40 can execute commands as any local user by providing the local user password.

```
udm_access    10.20.30.40,TS1004,tsup1004,allow,noauth
udm_access    10.20.30.40,TS1004,*,allow,auth
udm_access    10.20.30.40,*,*,deny,auth
udm_access    ALL,*,root,deny,auth
```

### Components

#### Universal Data Mover Server for IBM i

## 7.6.12 Universal Control Server for IBM i

The following set of rules permit services for the subnet 10.20.30 and denies all other connections unless an X.509 certificate is presented that maps to certificate ID operations.

```
uctl_access      10.20.30.,*,*,allow,auth
uctl_access      ALL,*,*,deny,auth

uctl_cert_access operations,*,allow,auth
uctl_cert_access *,*,deny,auth
```

When no certificate is presented that maps to a certificate ID, the following set of rules effectively permit connections from any host but has limited access from host 10.20.30.40 to user **TS1004** on that host. No host can execute commands as local user root. User **TS1004** on host 10.20.30.40 can execute commands as local user **tsup1004** without providing the password. Users **TS1004** from host 10.20.30.40 can execute commands as any local user by providing the local user password.

When a certificate is presented that maps to a certificate ID, certificate ID **joe** can request local user ID **tsup1004** without a password. Certificate ID **joe** is allowed to execute commands with any other local user ID with a password. Certificate ID operations cannot run anything. All other certificate IDs can execute commands with any user ID except for root with a password.

```
uctl_access      10.20.30.40,TS1004,tsup1004,allow,noauth
uctl_access      10.20.30.40,TS1004,*,allow,auth
uctl_access      10.20.30.40,*,*,deny,auth
uctl_access      ALL,*,root,deny,auth

uctl_cert_access joe,tsup1004,allow,noauth
uctl_cert_access joe,*,allow,auth
uctl_cert_access operations,*,deny,auth
uctl_cert_access *,root,deny,auth
```

## Components

### Universal Control

## 7.7 X.509 Certificates

A certificate is an electronic object that identifies an entity. It is analogous to a passport in that it must be issued by a party that is trusted by all who accept the certificate.

Certificates are issued by trusted parties called Certificate Authorities (CA's). For example, VeriSign Inc. is a CA that most parties trust. We all have faith that a trusted CA takes the necessary steps to confirm the identity of a user before issuing the user a certificate.

Certificate technology is based on public / private key technology. There are a few different types of public / private keys: RSA, DH, and DSS. As their name denotes, the private key must be kept private, like a password. The public key can be given to anyone or even published in a newspaper.

A property of public / private keys is that data encrypted with one can be decrypted only with the other. Therefore, if someone wants to send you a secret message, they encrypt the data with your public key, which everyone has. However, since you are the only one with your private key, you are the only one who can decrypt it. If you want to send someone message, such as a request for \$100,000 purchase, you can "sign" it with your private key.

**Note:** Signing does not encrypt the data. Once a person receives your request, that person can verify it is from you by verifying your electronic signature with your public key.

A certificate ties a statement of identity to a public key. Without the public key, the certificate is meaningless. Possession of a certificate alone does not prove your identity. You must have the corresponding private key. The two together prove your identity to any third party that trusts the CA that issued your certificate. This is a key point; if you do not trust the CA that signed a certificate, you cannot trust the certificate.

Since certificates originally were designed to be used for internet authentication, global directory technologies were developed to make them available via the internet. This directory technology is known as X.500 Directory Access Protocol. Later LDAP was introduced by Netscape to make it Lightweight Directory Access Protocol.

X.500 divides the world into a hierarchical directory. A person's identity is located by traversing down the hierarchy until it reaches the last node. Each node in the hierarchy consists of a type of object, such as a country, state, company, department, or name.

## 7.7.1 Sample Certificate Directory

Figure 7.8, below, provides a sample diagram of a small X.500 directory.

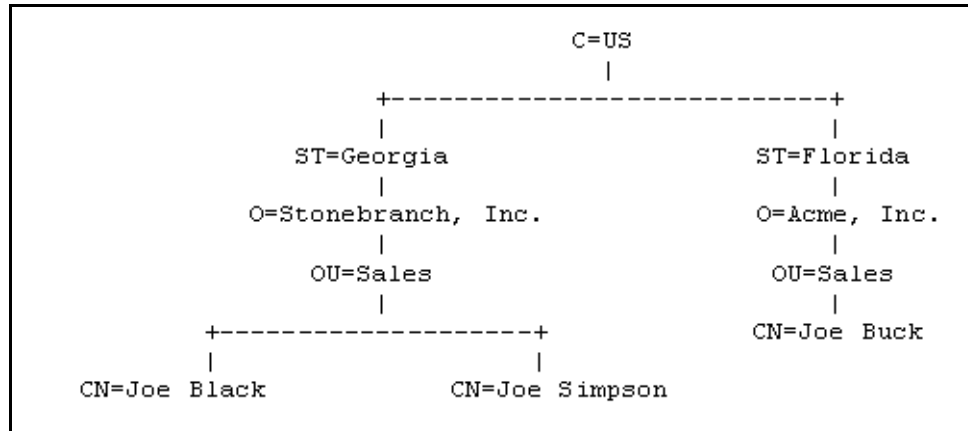


Figure 7.8 X.500 Directory (sample)

The keywords listed on each node are referred to as a Relative Distinguished Name (RDN). A person is identified by a Distinguished Name (DN). The DN value for Joe Black is **C=US/ST=Georgia/O=Stonebranch, Inc./OU=Sales/CN=Joe Black**.

A certificate is composed of many fields and possible extensions. Many of the most popular fields are specified as X.500 DN values.



## 7.7.2 Sample X.509 Certificate

Figure 7.9, below, illustrates a sample X.509 version 3 certificate for Joe Buck at the Acme corporation.

```

Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number:
      01:02:03:04:05:06:07:08
    Signature Algorithm: md5withRSAEncryption
    Issuer: C=US, ST=Florida, O=Acme, Inc., OU=Security, CN=CA
    Authority/emailAddress=ca@acme.com
    Validity
      Not Before: Aug 20 12:59:55 2004 GMT
      Not After : Aug 20 12:59:55 2005 GMT
    Subject: C=US, ST=Florida, O=Acme, Inc., OU=Sales, CN=Joe Buck
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      RSA Public Key: (1024 bit)
        Modulus (1024 bit):
          00:be:5e:6e:f8:2c:c7:8c:07:7e:f0:ab:a5:12:db:
          fc:5a:1e:27:ba:49:b0:2c:e1:cb:4b:05:f2:23:09:
          77:13:75:57:08:29:45:29:d0:db:8c:06:4b:c3:10:
          88:e1:ba:5e:6f:1e:c0:2e:42:82:2b:e4:fa:ba:bc:
          45:e9:98:f8:e9:00:84:60:53:a6:11:2e:18:39:6e:
          ad:76:3e:75:8d:1e:b1:b2:1e:07:97:7f:49:31:35:
          25:55:0a:28:11:20:a6:7d:85:76:f7:9f:c4:66:90:
          e6:2d:ce:73:45:66:be:56:aa:ee:93:ae:10:f9:ba:
          24:fe:38:d0:f0:23:d7:a1:3b
        Exponent: 65537 (0x10001)
    X509v3 extensions:
      X509v3 Basic Constraints:
        CA:FALSE
      X509v3 Subject Alternative Name:
        email:joe.buck@acme.com
    Signature Algorithm: md5withRSAEncryption
    a0:94:ca:f4:d5:4f:2d:da:a8:6d:e3:41:6e:51:83:57:b3:b5:
    31:95:32:b6:ca:7e:d1:4f:fb:01:82:db:23:a0:39:d8:69:71:
    31:9c:0a:3b:ce:f6:c6:e2:5c:af:23:f0:d7:ee:87:3e:8a:7b:
    40:03:39:64:a1:8c:29:7d:5b:99:93:fa:23:19:e1:e4:ac:4d:
    13:0f:de:ad:51:27:e3:4e:4b:9f:40:4c:05:fd:f2:82:09:3e:
    46:05:f0:ad:cc:f7:78:25:3e:11:f8:ca:b6:df:f7:37:57:9b:
    63:00:d0:b5:b5:18:ec:38:73:d2:85:a3:c7:24:21:47:ee:f2:
    8c:0d
  
```

Figure 7.9 X.509 Version 3 Certificate (sample)

**Note:** The contents of a certificate file does not look like the information in [Figure 7.9](#), which is produced by a certificate utility that uses the certificate file as input. Certificates can be saved in multiple file formats, so their file contents will look very different.

### 7.7.3 Certificate Fields

A certificate is composed of many fields.

[Table 7.5](#), below, describes the main certificate fields.

Field or Section	Description
Version	X.509 certificates come in two versions: 1 and 3.
Serial Number	CA is required to provide each certificate it issues a unique serial number. The serial number is not unique for all certificates, only for the certificates issued by each CA.
Issuer	DN name of the CA that issued the certificate.
Validity	Starting and ending date for which this certificate is valid.
Subject	Identity of the certificate. A certificate may identify a person or a computer. In this case, the certificate identifies Joe Buck in the Sales organization of the Acme company in the state of Florida in the United States.
Public Key	Public key associated with the certificate identity.
X509v3 Extensions	X.509 version 3 introduced this section so that additional certificate fields may be added. In this case, the identity's email address is included as a Subject Alternative Name field. This section is not available in X.509 version 1.
Signature	CA's digital signature of the certificate.

Table 7.5 Certificate Fields

---

## 7.7.4 SSL Peer Authentication

---

The SSL protocol utilizes X.509 certificates to perform peer authentication. For example, a Universal Data Mover Manager may want to authenticate that it is connected to the correct Broker.

Peer authentication is performed by either one or both of the programs involved in the network session. If a Manager wishes to authenticate the Broker to which it connects, the Broker will send its certificate to the Manager for the Manager to authenticate. Should the Broker wish to authenticate the Manager, the Manager sends its certificate to the Broker.

Certificate authentication is performed in the following steps:

1. Check that the peer certificate is issued by a trusted CA.
2. Check that the certificate has not been revoked by the CA.
3. Check that the certificate identifies the intended peer.

If a step fails, the network session is terminated immediately.

### Certificate Verification

The Stonebranch Solutions component must be configured with a list of trusted CA certificates. When a peer certificate is received, the trusted CA certificates are used to verify that the peer certificate is issued by one of the trusted CA's.

The trusted CA certificate list must be properly secured so that only authorized accounts have update access to the list. Should the trusted CA list become compromised, there is a possibility that an untrusted CA certificate was added to the list.

The CA certificate list configuration option is `CA_CERTIFICATES`. It specifies a PEM formatted file that contains one or more CA certificates used for verification.

Should a peer certificate not be signed by a trusted CA, the session is immediately terminated.

### Certificate Revocation

After a certificate is verified to have come from a trusted CA, the next step is to check if the CA has revoked the certificate. Since a certificate is held by the entity for which it identifies, a CA cannot take a certificate back after it is issued. So when a CA needs to revoke a certificate for some reason, it issues a list of revoked certificates referred to as the Certificate Revocation List (CRL). A program that validates certificates needs to have access to the latest CRL issued by the CA.

The `CERTIFICATE_REVOCATION_LIST` configuration option specifies the PEM formatted file that contains the CRL. This option is available in all Stonebranch Solutions components that utilize certificates.

## Certificate Identification

Once a certificate is validated as being issued by a trusted CA, and not revoked by the CA, the next step is to check that it identifies the intended peer.

A Stonebranch Solutions Manager validates a Broker certificate by the Broker host name or IP address or the certificate serial number. The `VERIFY_HOST_NAME` configuration option is used to specify the host name or IP address that is identified in the Broker certificate. Each certificate signed by a CA must have a unique serial number for that CA. The `VERIFY_SERIAL_NUMBER` option is used to specify the serial number in the Broker certificate.

Should certificate identification fail, the session is immediately terminated.

Universal Brokers work differently than the Managers. A Broker maps a peer certificate to a certificate ID. The certificate map definitions are part of the Universal Access Control List (UACL) definitions. At that point, the certificate ID is used by UACL definitions to control access to Broker and Server services.

## Certificate Support

Many certificate authority applications, also known as Public Key Infrastructure (PKI) applications, are available. Stonebranch Solutions should be able to utilize any certificate in a PEM format file. PEM (Privacy Enhanced Mail) is a common text file format used for certificates, private keys, and CA lists.

Stonebranch Solutions support X.509 version 1 and version 3 certificates.

Although implementing a fully featured PKI infrastructure is beyond the scope of Stonebranch Solutions and this documentation, some assistance is provided using the OpenSSL toolkit (<http://www.openssl.org>).

Stonebranch Solutions on most of the supported platforms utilize the OpenSSL toolkit for its SSL and certificate implementation. OpenSSL is delivered on most UNIX distributions and Windows distributions are available on the OpenSSL website.

Stonebranch Solutions supports z/OS System SSL on the IBM z/OS operating system as well as OpenSSL. System SSL interfaces directly with the RACF security product for certificate access. All certificates, CA and user certificates, and private keys must be stored in the RACF database to use System SSL.

The Stonebranch Solutions suite includes an X.509 certificate utility, Universal Certificate, to create certificates for use in the Stonebranch Solutions suite. See Chapter 2 [Universal Certificate](#) in the [Stonebranch Solutions Utilities Reference Guide](#) for details.

---

## 7.8 Creating Certificates Examples

This section provides examples that illustrate how to use Universal Certificate.

The examples provide the command line options only so that they can be used easily in any environment.

[Creating a Certificate Authority Certificate](#)

[Creating a Certificate](#)

## 7.8.1 Creating a Certificate Authority Certificate

The first step in creating a certificate hierarchy is creating the root Certificate Authority (CA) certificate. The CA certificate is used to issue user certificates.

A certificate is created by creating a certificate request and then having the CA validate and sign the certificate. Since we are creating a root CA certificate, there is no CA to sign the certificate request, so instead a self-signed certificate is created and the CA flag is set.

The following command creates:

- Certificate request, which it writes to file `req.pem`
- Private key, which it writes to file `cakey.pem`

```
-create request -request_file req.pem -private_key_file cakey.pem  
-country US -state Maryland -locality Baltimore -organization "Acme, Inc."  
-common_name "Acme CA"
```

It is imperative that the private key file `cakey.pem` is secured so that no one other than the CA has read access. If unauthorized access is gained to the CA's private key, all certificates issued by the CA no longer can be trusted.

The following command creates the CA certificate and writes it to file `cacert.pem`.

```
-create cert -request_file req.pem -cert_file cacert.pem  
-private_key_file cakey.pem -ca yes
```

The CA certificate, `cacert.pem`, must be made available to any system that wants to consider the certificates issued by the CA as valid.

### Components

#### Universal Certificate

## 7.8.2 Creating a Certificate

---

There are two steps in creating a certificate:

- First step is performed by the party that wants the certificate.
- Second step is performed by the Certificate Authority (CA) that creates the certificate.

### Step 1

Step one is creating the certificate request. The certificate request will then be sent to the CA that verifies the request and creates the certificate from the request. The command that creates the certificate request also creates a private key. The private key must be secured so that only the entity identified by the certificate request has read access.

The following command creates:

- Certificate request, which it writes it to file `req.pem`.
- Private key, which it writes it to file `pkey.pem`.

```
-create request -request_file req.pem -private_key_file pkey.pem -country US  
-state Maryland -locality Baltimore -organization "Acme, Inc."  
-common_name "Joe Buck"
```

### Step 2

Step two is for the CA to create a certificate from the request and sign it with the CA's private key.

The following command creates the certificate and writes it to file `cert.pem`.

```
-create cert -request_file req.pem -cert_file cert.pem  
-private_key_file cakey.pem -ca_cert_file cacert.pem
```

## Components

### Universal Certificate

---

# Configuration Management

---

## 8.1 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 product configuration, there are multiple methods available to configure the products in order to meet your needs.



## 8.2 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 Stonebranch Solutions 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 Line File](#)
3. [Environment Variables](#)
4. [Configuration File](#)

The command line, command line 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 options and preferences for all executions of a component.

This order of precedence means that a command option specified on the command line overrides the same option specified in a command line 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 Stonebranch Solutions servers, are configurable only by modifying their configuration files (see [Section 8.2.4 Configuration File](#)). They are not configurable via command line, command line file, or environmental variables.

## 8.2.1 Command Line

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

Command line options are the highest in order of precedence of all the configuration methods (see Section [8.2 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.

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

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

### z/OS

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

Command line options are prefixed with a dash ( - ) character. For many options, there are two different forms in which they can be specified:

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

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

Example command line options specified in the PARM value follow:

#### Short form:

```
PARM='-I 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 and Windows

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

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

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

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

Example command line options follow:

#### Short form:

```
-l info -G yes
```

#### Long form:

```
-level info -login yes  
-LEVEL info -LoGiN YES
```

### IBM i

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

Example command line options follow:

Command line options:

```
MSGLEVEL(INFO) COMPRESS(*YES)
```

All of the Stonebranch Inc. Stonebranch Solutions 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.

---

## 8.2.2 Command Line File

---

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

The command line file is the second to highest in the precedence order, after command line options (see Section [8.2 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 line file can be encrypted if it is necessary to secure the contents.

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

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

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

## 8.2.3 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 line file options (see Section [8.2 Configuration Methods](#)).

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

All environment variables used by Stonebranch Solutions 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 by a called program.

Example of setting an environment variable:

```
Set option UCMDLEVEL to a value of INFO in a bourne, bash, or korn shell:  
UCMDLEVEL=INFO  
export UCMDLEVEL
```

### Windows

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

Example of setting an environment variable:

Set option UCMDLEVEL to a value of INFO:

```
SET UCMDLEVEL=INFO
```

### IBM i

Environment variables in IBM i are defined with Command Language (CL) commands for the current job environment.

Example of setting an environment variable:

Set option UCMDLEVEL to a value of INFO:

```
ADDENVVAR ENVVAR(UCMDLEVEL) VALUE(INFO)
```

## 8.2.4 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 Section [8.2 Configuration Methods](#)).

(For most Stonebranch Solutions components, some options can be specified only in a configuration file, while other options can be overridden by individual command executions. The Stonebranch Solutions reference guide for each component identifies these options.)

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

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

Universal Broker reads the configuration files when it first starts or when it receives a configuration refresh request from Universal Control or Universal Enterprise Controller. Any changes made to a configuration file are not in effect until the Broker is recycled or receives a configuration refresh request (see Section [8.5 Configuration Refresh](#)).

Universal Broker can operate in managed or unmanaged mode:

- In unmanaged mode, the configuration information for the various Stonebranch Solutions 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 Stonebranch Solutions components is "locked down" and can be modified or viewed only via the I-Management Console.

(For detailed information on unmanaged and managed modes, see Section [8.3 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](#) for the configuration file syntax.

## UNIX

Configuration files are regular text files on UNIX.

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:

Directory	Notes
/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	

Table 8.1 UNIX Configuration File Directory Search

See [Configuration File Syntax](#) for the configuration file syntax.

## 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.

The 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 (see [Section 8.4 Universal Configuration Manager](#)).

## IBM i

The configuration files on IBM i are stored in a source physical file named UNVCONF in the UNVPRD420 library. The files can be edited with a text editor.

See [Configuration File Syntax](#) for the configuration file syntax.



## Configuration File Syntax

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

The following rules apply for configuration file syntax:

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

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

## 8.3 Remote Configuration

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

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

Universal Broker supports remote configuration in either of two modes:

- [Unmanaged Mode](#)
- [Managed Mode](#)

### 8.3.1 Unmanaged Mode

---

Unmanaged mode is the default mode of operations for Universal Broker. It allows a Universal Broker – and the Stonebranch Solutions 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 Stonebranch Solutions 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 Stonebranch Solutions component, the Universal Broker updates the appropriate configuration file. The component will use this new configuration at its next invocation.

**Note:** If errors or invalid configuration values are updated via I-Management Console for a component other than Universal Broker, the component may not run successfully until the configuration has been corrected.

---

## 8.3.2 Managed Mode

---

When a Universal Broker is operating in managed mode, the configuration information for all Stonebranch Solutions 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 Stonebranch Solutions 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 the [Universal Enterprise Controller Client Applications User Guide](#) for specific information on how to select managed mode.)

Figure 8.1, below, illustrates remote configuration for one Agent in managed mode and one Agent in unmanaged mode.

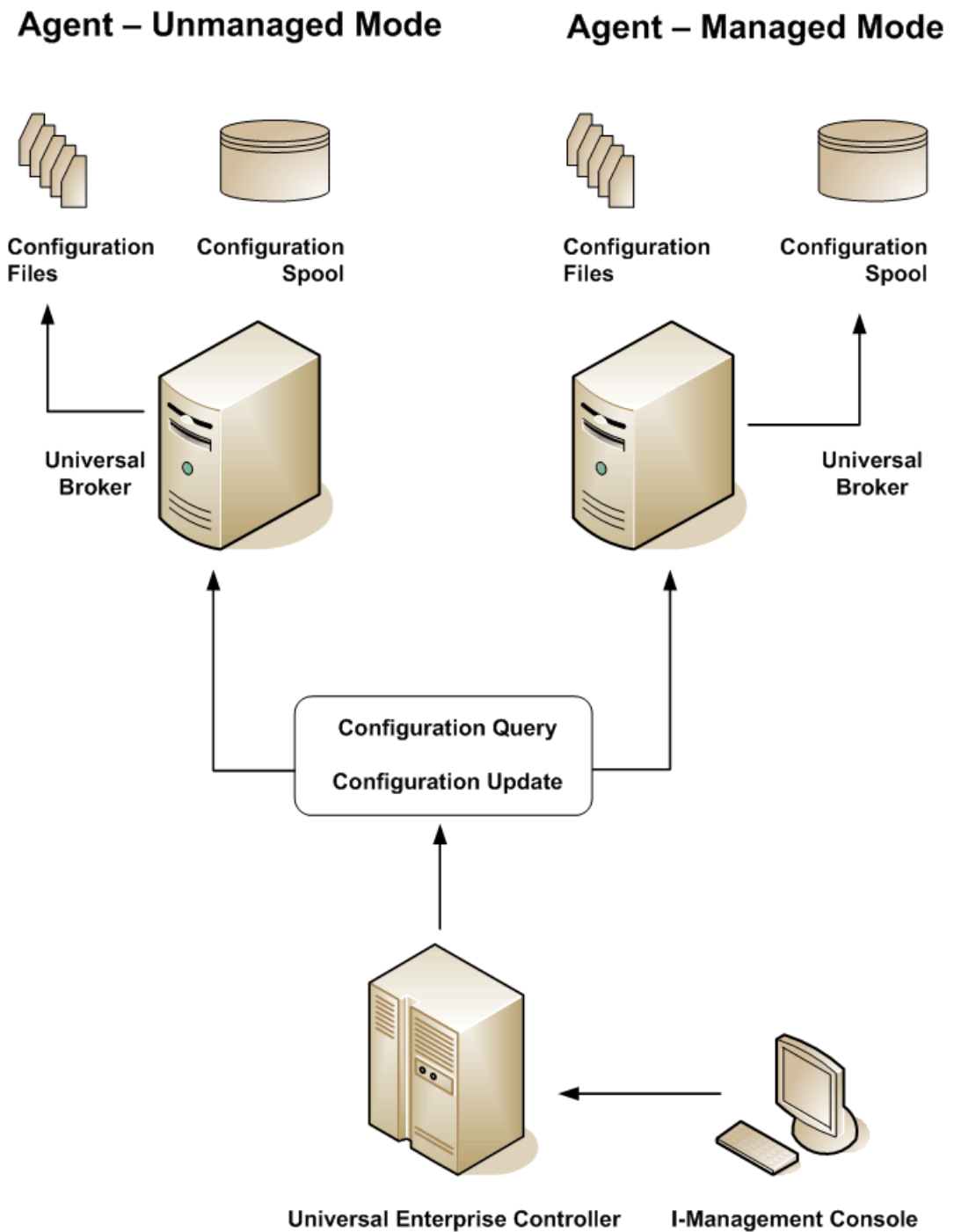


Figure 8.1 Remote Configuration - Unmanaged and Managed Modes of Operation

---

### 8.3.3 Universal Broker Start-up

---

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

#### *Unmanaged Mode*

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

#### *Managed Mode*

At Universal Broker start-up in managed mode, the Universal Broker configuration file points Universal Broker to the location of the configuration spool file, from which the Broker retrieves configuration information for all Stonebranch Solutions 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.

## 8.4 Universal Configuration Manager

The Universal Configuration Manager is a Stonebranch Solutions graphical user interface application that enables you to configure all of the Stonebranch Solutions 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.

### 8.4.1 Availability

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

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

#### Windows Vista, Windows 7

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

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

1. Universal Configuration Manager may issue the following error:

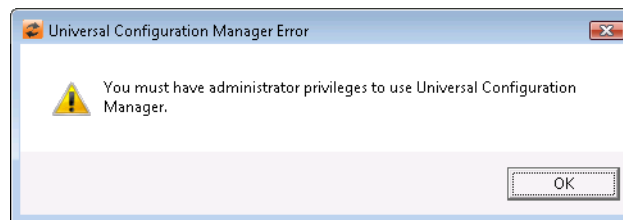


Figure 8.2 Universal Configuration Manager Error dialog – Windows Vista / Windows 7

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

The Windows Vista / Windows 7 Program Compatibility Assistant (PCA) displays the following dialog:

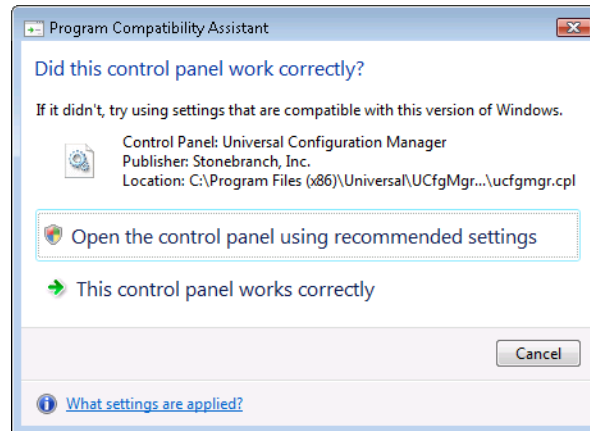


Figure 8.3 Program Compatibility Assistant – Windows Vista / Windows 7

3. To continue, select **Open the control panel using recommended settings**. This instructs the PCA to "shim" (Microsoft term) the Configuration Manager, establishing it as an application that requires elevated privileges.  
Windows Vista / Windows 7 User Account Control (UAC) then displays a prompt seeking permission to elevate the logged-in account's access token.
4. Select **Continue** to give the account full administrative privileges.  
Subsequent attempts to open Universal Configuration Manager should result only in the UAC prompt.

## 8.4.2 Accessing the Universal Configuration Manager

---

To access the Universal Configuration Manager:

1. Click the **Start** icon at the lower left corner of your Windows operating system screen to display the Start menu.
2. Click (Settings/) Control Panel on the Start menu to display the Control Panel screen.
3. Select the Universal Configuration Manager icon to display the Universal Configuration Manager screen (see [Figure 8.4](#)).

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

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

For example, the Windows XP Category View places the Universal Configuration Manager icon under the **Other Control Panel Options** link. Windows Vista, Windows 7, and Windows Server 2008 / 2008 R2 place the icon within the **Additional Options** category.

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

### 64-bit Windows Editions

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

When using the Category View, look for the 32-bit Control Panel applet icons in the **Additional Options** category.



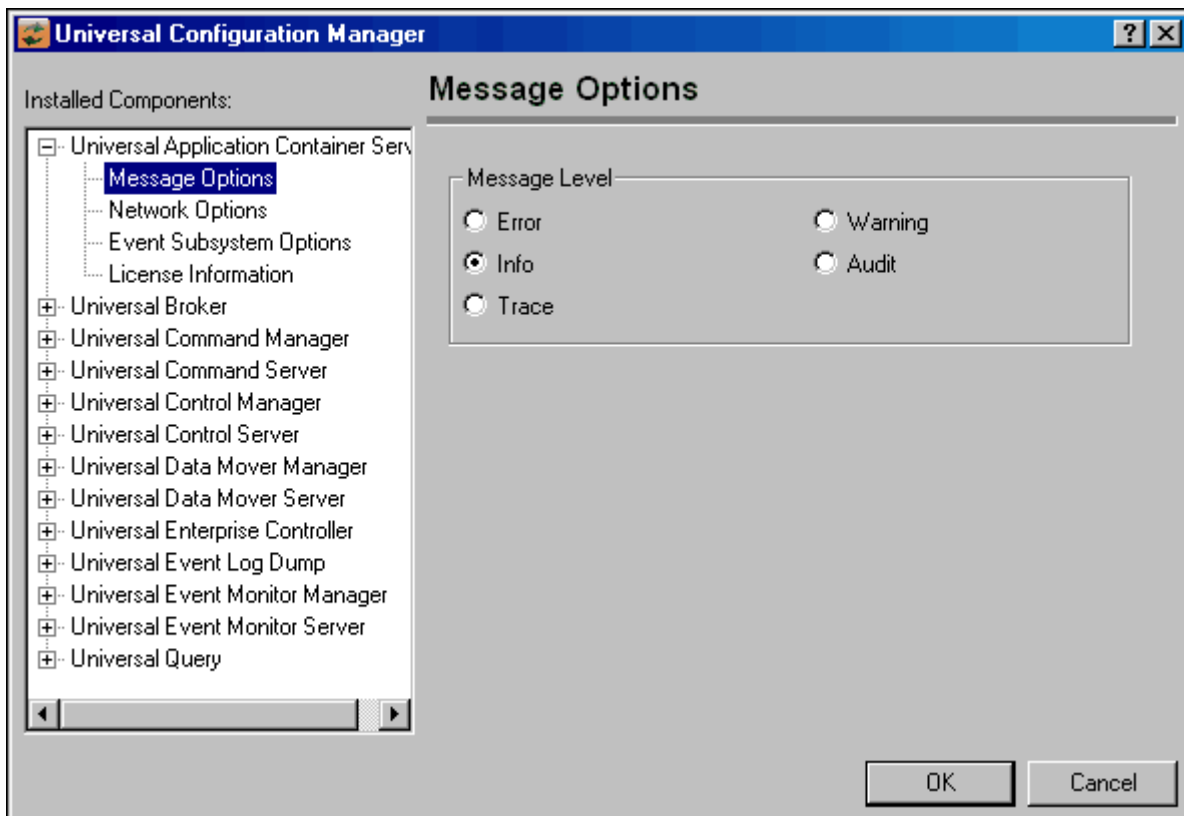


Figure 8.4 Universal Configuration Manager

Each Universal Configuration Manager screen contains two sections:

1. Left side of the screen displays the Installed Components tree, which lists:
  - Stonebranch Solutions 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.)

---

## 8.4.3 Navigating through Universal Configuration Manager

---

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

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

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

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

In [Figure 8.4](#), for example:

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

---

## 8.4.4 Modifying / Entering Data

---

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

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

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

**Note:** You do not have to click the **OK** button after every modification or entry, or on every property page on which you have modified and/or entered data. Clicking **OK** just once, on any page, will save the modifications and entries made on all pages – and will exit Universal Configuration Manager (see [Section 8.4.5 Saving Data.](#))

### Rules for Modifying / Entering Data

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

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

---

## 8.4.5 Saving Data

---

To save all of the modifications / entries made on all of the property pages, click the **OK** button at the bottom of any property page. The information is saved in the configuration 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.

---

## 8.4.6 Accessing Help Information

---

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

To access Help:

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

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

The Universal Configuration Manager's context-sensitive help is a WinHelp file, which Windows Vista, Windows 7, and Windows Server 2008 / 2008 R2 do not support.

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

## 8.4.7 Universal Data Mover Installed Components

### Universal Data Mover Manager

Figure 8.5 illustrates the Universal Configuration Manager screen for the Universal Data Mover Manager.

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

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

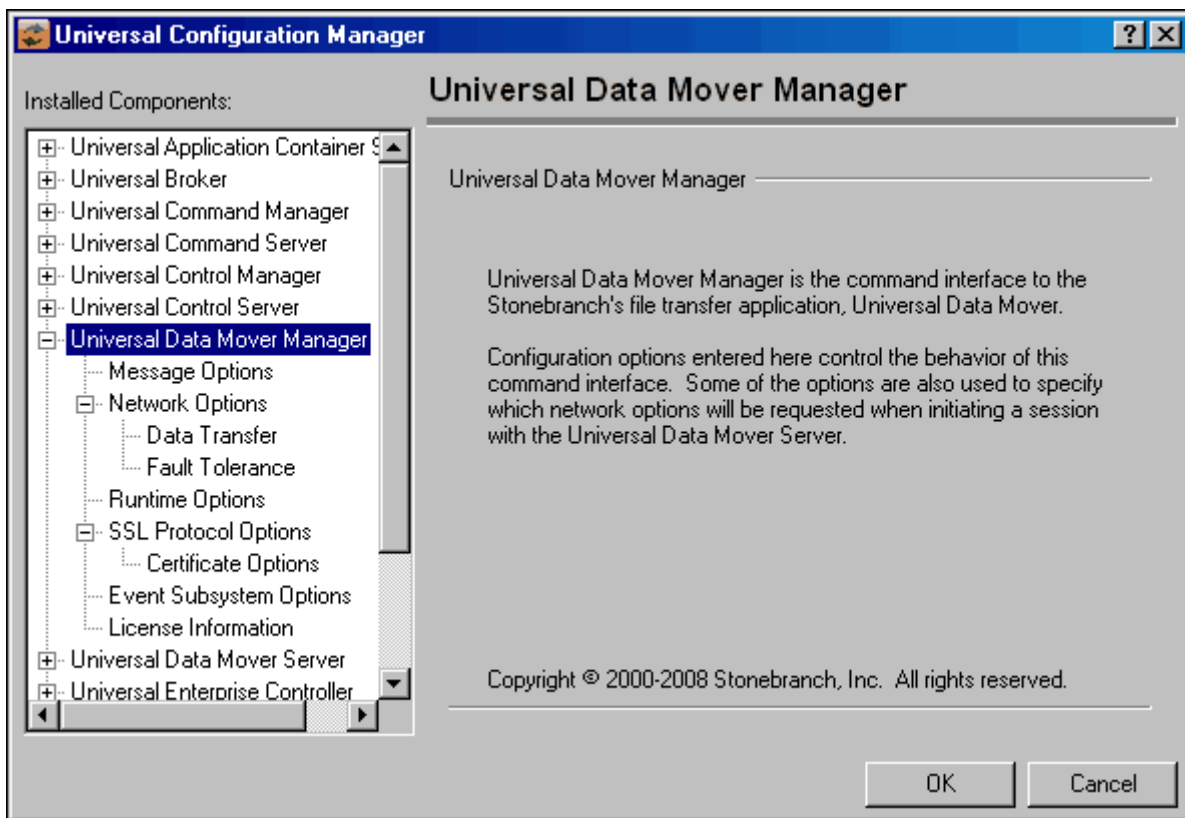


Figure 8.5 Universal Configuration Manager - UDM Manager

## Universal Data Mover Server

Figure 8.6 illustrates the Universal Configuration Manager screen for the Universal Data Mover Server.

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

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

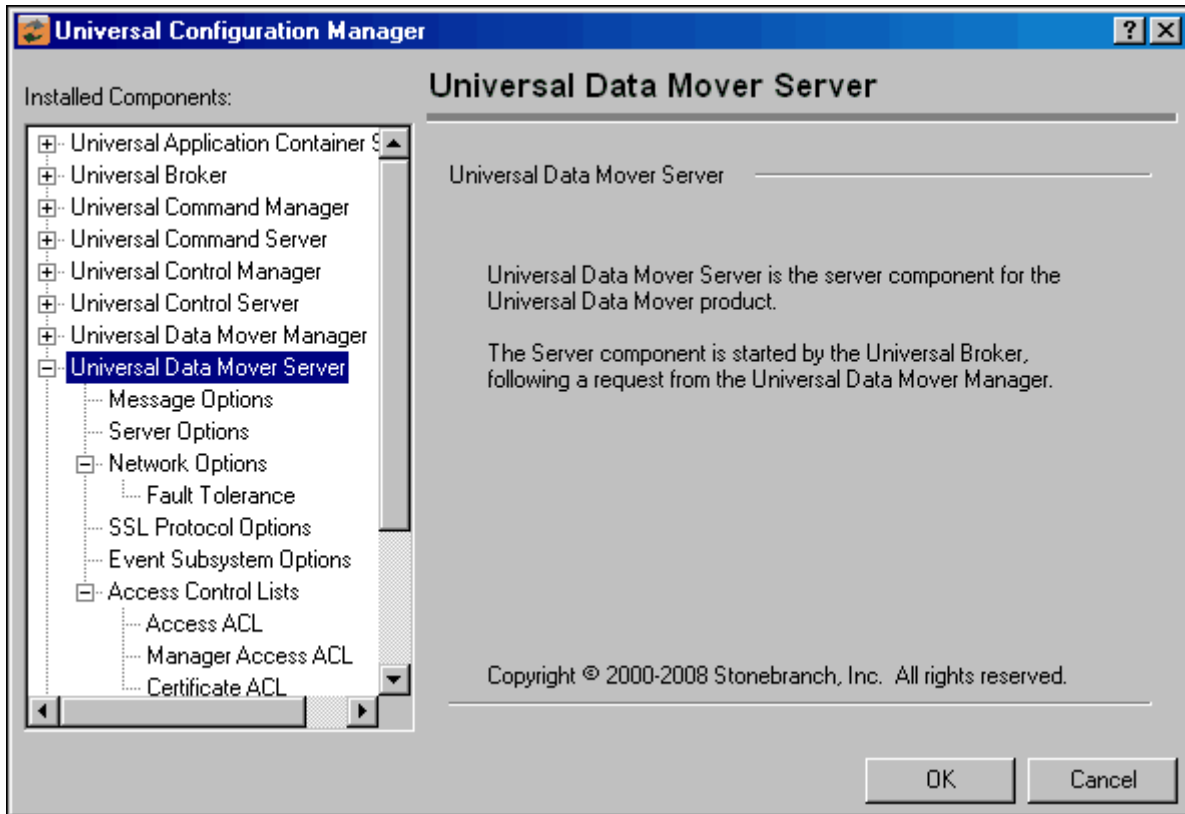


Figure 8.6 Universal Configuration Manager - UDM Server

## 8.4.8 Universal Event Monitor Installed Components

### Universal Event Monitor Manager

Figure 8.7 illustrates the Universal Configuration Manager screen for the Universal Event Monitor Manager.

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

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

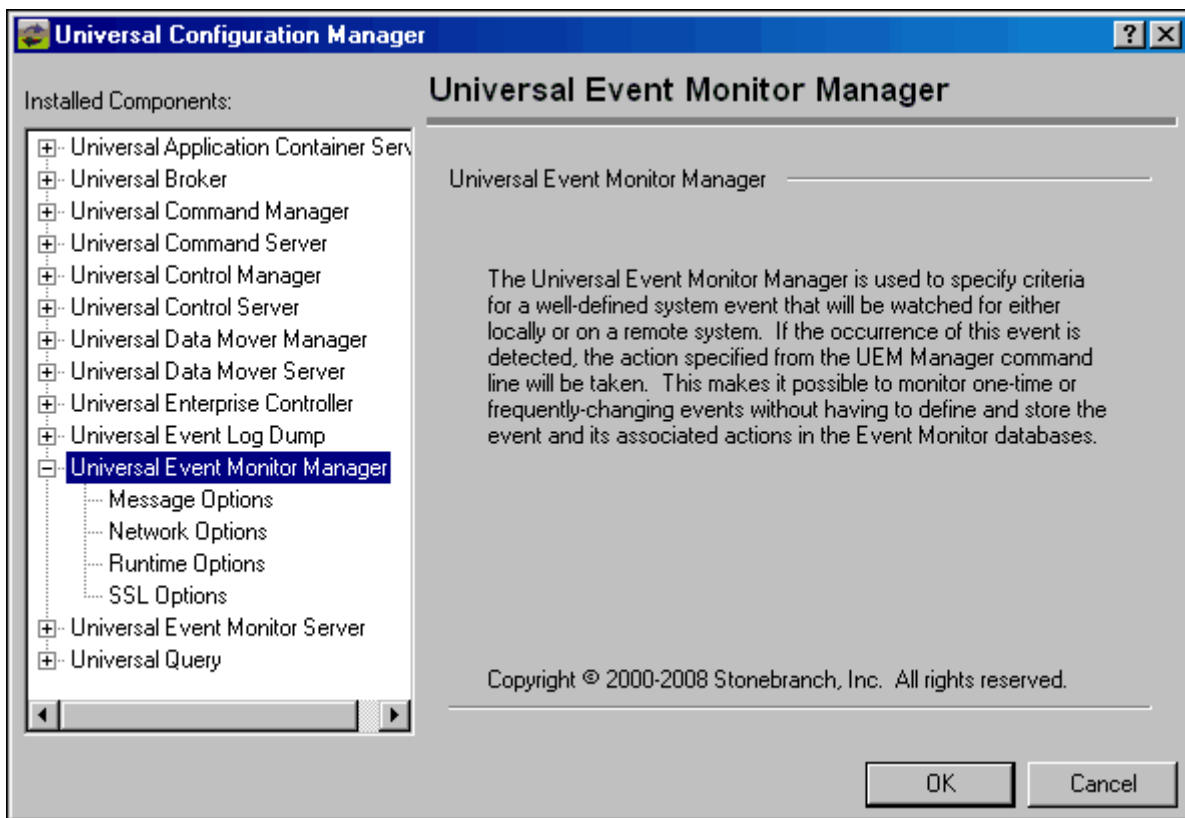


Figure 8.7 Universal Configuration Manager - UEM Manager

## Universal Event Monitor Server

Figure 8.8, illustrates the Universal Configuration Manager screen for the Universal Event Monitor Server.

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

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

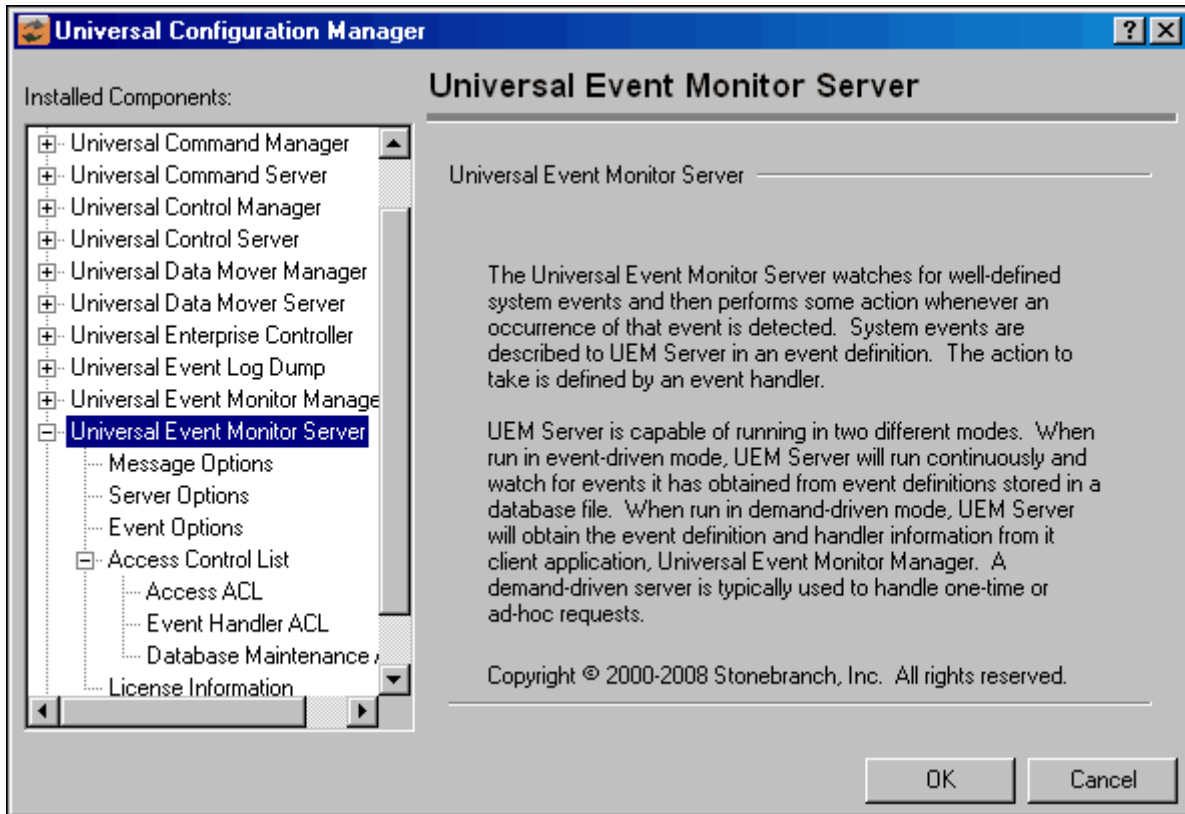


Figure 8.8 Universal Configuration Manager - UEM Server

## 8.4.9 Universal Enterprise Controller Component

Figure 8.9 illustrates the Universal Configuration Manager screen for the Universal Enterprise Controller.

The Installed Components list identifies all of the UEC property pages.

The text describes the selected component, Universal Enterprise Controller.

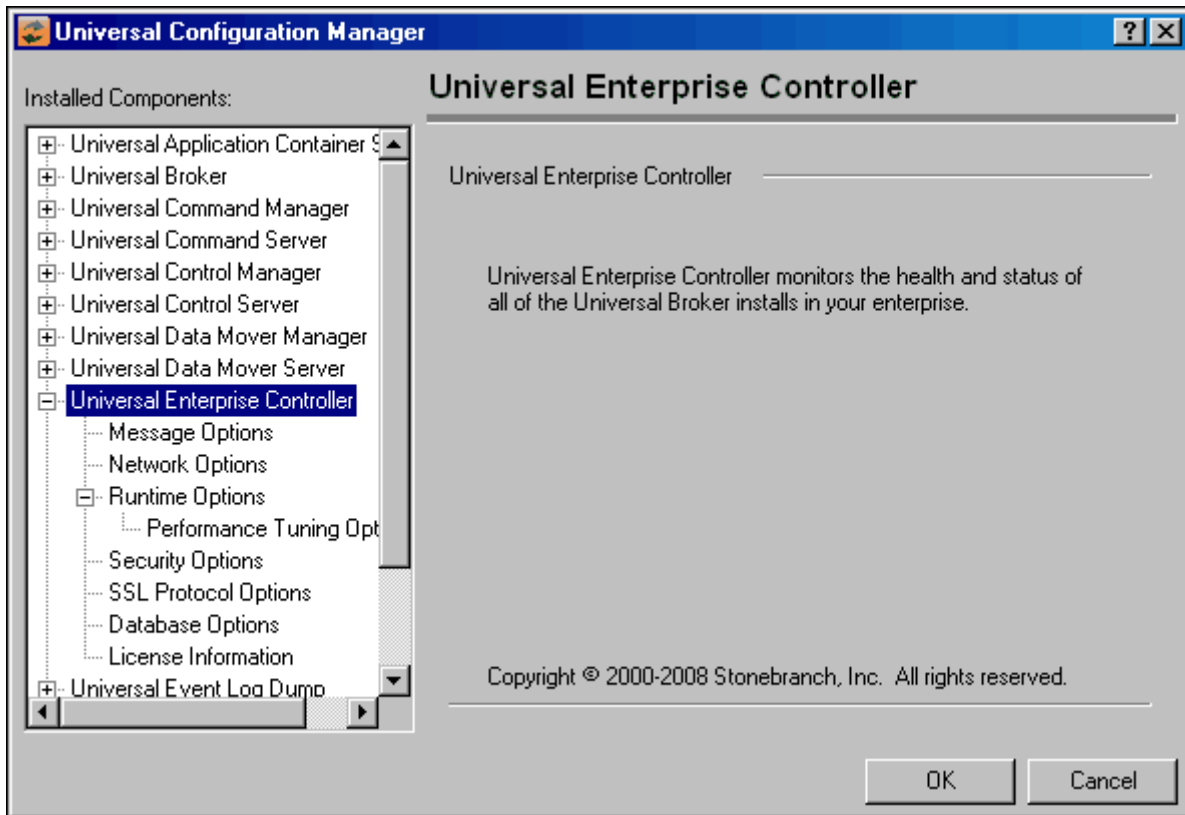


Figure 8.9 Universal Configuration Manager - Universal Enterprise Controller



## 8.4.10 Universal Broker Installed Component

Figure 8.10 illustrates the Universal Configuration Manager screen for the Universal Broker.

The Installed Components list identifies all of the Universal Broker property pages. The text describes the selected component, Universal Broker.

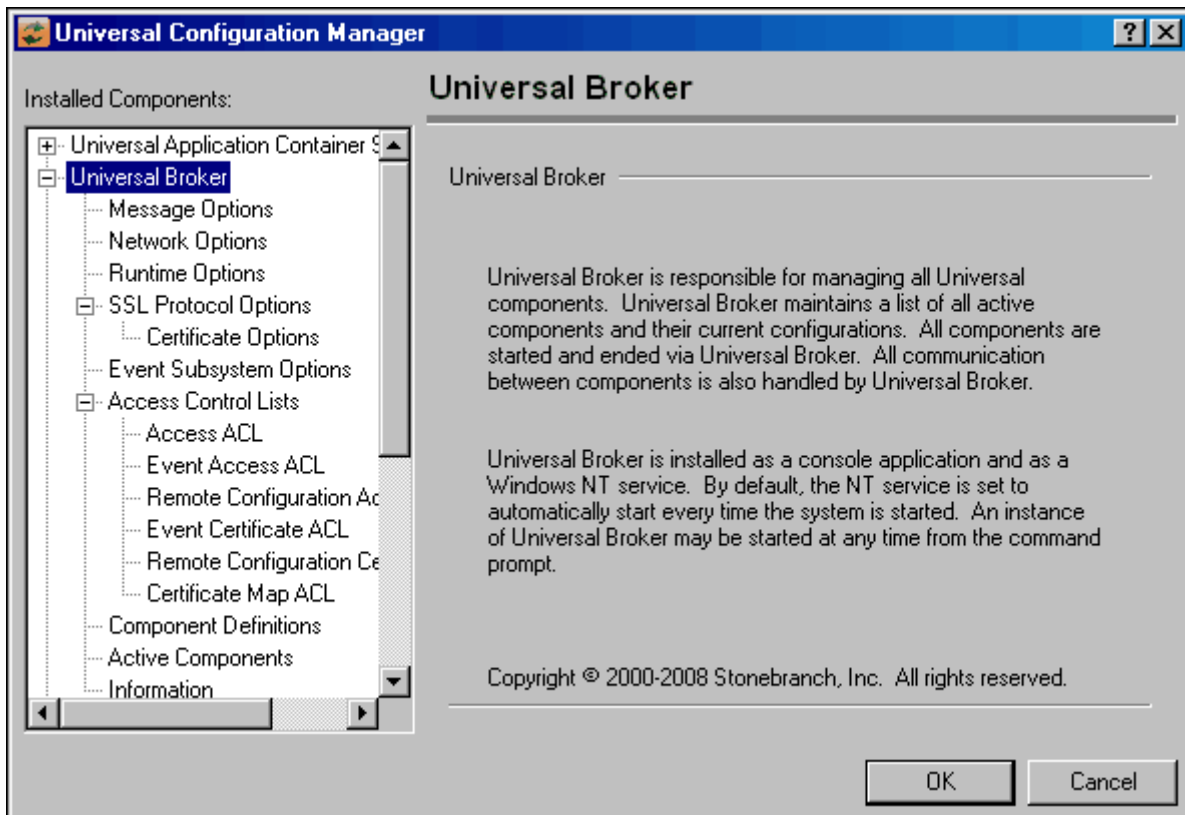


Figure 8.10 Universal Configuration Manager - Universal Broker

## 8.5 Configuration Refresh

Universal Broker maintains the configuration files for all Stonebranch Solutions components that it manages. The components do not read their configuration files themselves (except for Universal Enterprise Controller, which does read its own configuration file).

When a component starts, it first registers with its local Universal Broker. As part of the registration process, Universal Broker returns the configuration data to the component.

Universal Broker reads the configuration files at initial start-up and, thereafter, whenever it is refreshed; that is, when either of the following occurs:

- Universal Broker is recycled (stopped and restarted).
- Universal Broker is refreshed by Universal Control.
- Universal Broker is refreshed by Universal Enterprise Controller (via I-Management Console).

### Windows

- Universal Broker is refreshed by Universal Configuration Manager.

After a configuration file has been modified, the Universal Broker must be refreshed in order for the modified values to take effect. Refreshing a Universal Broker directs it to read its configuration data and update its current configuration settings.

## 8.5.1 Configuration Refresh Via Universal Control

---

Universal Control refreshes the Universal Broker by issuing a configuration refresh request via its [REFRESH\\_CMD](#) configuration option.

Universal Control directs Universal Broker to refresh the configuration data of all components, including itself, or a single component. (Currently, the only individual component can be refreshed this way is the Universal Event Monitor Server.)

### Configuration Refresh for Universal Event Monitor Server

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

When a change is made to the stored UEM Server configuration settings (see Section [8.2.4 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 Section [8.6 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 (see Section [8.4 Universal Configuration Manager](#)).

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

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

## 8.5.2 Configuration Refresh Via Universal Configuration Manager

When any of the options that can be refreshed are updated using the [Universal Configuration Manager](#), a configuration refresh request is sent to Universal Broker, and its configuration is refreshed automatically.

The configuration refresh request directs Universal Broker to take the following actions:

Step	Description
Step 1	Read its configuration file. Universal Broker refreshes its configuration options.
Step 2	Read all component definitions found in the component definition directory. The Broker replaces all component definitions with the newly read component definitions. New component definitions are added and deleted component definitions are removed.
Step 3	Read the Universal Access Control List configuration entries from the registry. The Broker replaces its UACL entries with the newly read entries.

## 8.5.3 Universal Broker Configuration Options Refresh

As with all Stonebranch Solutions components, all Universal Broker options can be modified by editing the configuration file directly. However, unlike other components, not all Universal Broker options can be modified via I-Management Console. In I-Management Console, these Universal Broker options are read-only.

Some Universal Broker options can be modified only by editing the Universal Broker configuration file, `ubroker.conf`. For these modifications to take effect, Universal Broker must be recycled.

All other Universal Broker options can be modified either:

- By editing `ubroker.conf`.
- Via I-Management Console.
- Via the [Universal Configuration Manager](#).

Depending on the option, for a modification to take effect:

- Universal Broker must be recycled.
- Universal Broker must be refreshed by issuing a Universal Control configuration refresh request (via the `REFRESH_CMD` configuration option), if the modifications are made in `ubroker.conf`.
- Universal Broker is refreshed automatically, if the modifications are made via I-Management Console or the [Universal Configuration Manager](#).

For a list of the Universal Broker configuration options in each category, see the Universal Broker Reference Guide, Chapter [10 Universal Broker Configuration Options Refresh](#).

---

## 8.6 Refreshing via Universal Control Examples

This section provides examples of how to refresh configuration data of all components, including itself, or a single component, using a Universal Control. (Currently, the only individual component that can be refreshed is the Universal Event Monitor Server.)

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

---

### z/OS

[Refreshing Universal Broker from z/OS](#)

[Refreshing a Component from z/OS](#)

---

### Windows

[Refreshing Universal Broker via Universal Control from Windows](#)

[Refreshing a Component via Universal Control from Windows](#)

---

### UNIX

[Refreshing Universal Broker via Universal Control from UNIX](#)

[Refreshing a Component via Universal Control from UNIX](#)

---

### IBM i

[Refreshing Universal Broker via Universal Control from IBM i](#)

[Refreshing a Component via Universal Control from IBM i](#)

## 8.6.1 Refreshing Universal Broker from z/OS

The z/OS version of the Universal Control configuration refresh request is:

```
//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
//
```

Figure 8.11 Refreshing Universal Broker via Universal Control from z/OS

This example refreshes the Broker configuration on host **dallas**.

The REFRESH command directs the Broker to take the following actions:

Step	Procedure
Step 1	Read its configuration file. The Broker refreshes configuration options.
Step 2	Read all component definitions found in ddname <b>UNVCONF</b> . The Broker replaces all component definitions with the newly read component definitions. New component definitions are added and deleted component definitions are removed.
Step 3	Read the Universal Access Control List configuration file allocated to ddname <b>UNVACL</b> . The Broker replaces its UACL entries with the newly read entries.

## SYSIN Options

The SYSIN options used in this example are:

Option	Description
<code>-refresh</code>	Instruction to refresh Universal Broker on the remote system.
<code>-host</code>	Directs the command to a computer with a host name of <b>dallas</b> .

## Components

### Universal Control

## 8.6.2 Refreshing a Component from z/OS

This example refreshes a component on a remote system.

```
//jobname JOB CLASS=A,MSGCLASS=X
//STEP1 EXEC UCTLPRC
//SYSIN DD *
-refresh uems -cmdid "UEM-dallas" -host dallas -userid joe -pwd akksdiq
/*
```

Figure 8.12 Refreshing Component via Universal Control from z/OS

### SYSIN Options

The SYSIN options used in this example are:

Option	Description
<code>-refresh</code>	Type of component to refresh on the remote system.
<code>-cmdid</code>	Assigns a command identifier of "UEM-dallas" to the started component.
<code>-host</code>	Directs the command to a computer with a host name of <b>dallas</b> .
<code>-userid</code>	Remote user ID with which to execute the Universal Control Server process.
<code>-pwd</code>	Password for the user ID.

### Components

#### Universal Control



## 8.6.3 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
```

Figure 8.13 Refreshing Universal Broker via Universal Control from Windows

### Command Line Options

The command line options used in this example are:

Option	Description
<code>-refresh</code>	Instruction to refresh Universal Broker on the remote system.
<code>-host</code>	Directs the command to a computer with a host name of <b>dallas</b> .
<code>-userid</code>	Remote user ID with which to execute the Universal Control Server process.
<code>-pwd</code>	Password for the user ID.

This refresh request directs Universal Broker to take the following actions:

Step	Description
Step 1	Read its configuration file. Universal Broker refreshes its configuration options.
Step 2	Read all component definitions found in the component definition directory. The Broker replaces all component definitions with the newly read component definitions. New component definitions are added and deleted component definitions are removed.
Step 3	Read the Universal Access Control List configuration entries from the registry. The Broker replaces its UACL entries with the newly read entries.

### Components

#### Universal Control

## 8.6.4 Refreshing a Component via Universal Control from Windows

This example refreshes a component on a remote system.

```
uctl -refresh uems -cmdid "UEM-dallas" -host dallas -userid joe -pwd akksdiq
```

Figure 8.14 Refreshing Component via Universal Control from Windows

### Command Line Options

The command line options used in this example are:

Option	Description
-refresh	Type of component to refresh on the remote system.
-cmdid	Assigns a command identifier of "UEM-dallas" to the started component.
-host	Directs the command to a computer with a host name of dallas.
-userid	Remote user ID with which to execute the Universal Control Server process.
-pwd	Password for the user ID.

### Components

#### Universal Control

## 8.6.5 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
```

Figure 8.15 Refreshing Universal Broker via Universal Control from UNIX

### Command Line Options

The command line options used in this example are:

Option	Description
<code>-refresh</code>	Instruction to refresh Universal Broker on the remote system.
<code>-host</code>	Directs the command to a computer with a host name of <code>dallas</code> .
<code>-userid</code>	Remote user ID with which to execute the Universal Control Server process.
<code>-pwd</code>	Password for the user ID.

This refresh request directs Universal Broker to take the following actions:

Step	Description
Step 1	Read its configuration file <code>ubroker.conf</code> . Universal Broker refreshes the following configuration options: <ul style="list-style-type: none"> <li><code>MESSAGE_LANGUAGE</code></li> <li><code>RUNNING_MAX</code></li> </ul>
Step 2	Read all component definitions found in the component definition directory. The Broker replaces all component definitions with the newly read component definitions. New component definitions are added and deleted component definitions are removed.
Step 3	Read the Universal Access Control List configuration file <code>uac1.conf</code> . The Broker replaces its UACL entries with the newly read entries.

### Components

#### Universal Control

## 8.6.6 Refreshing a Component via Universal Control from UNIX

This example refreshes a component on a remote system.

```
uctl -refresh uems -cmdid "UEM-dallas" -host dallas -userid joe -pwd akksdiq
```

Figure 8.16 Refreshing Component via Universal Control from UNIX

### Command Line Options

The command line options used in this example are:

Option	Description
-refresh	Type of component to refresh on the remote system.
-cmdid	Assigns a command identifier of "UEM-dallas" to the started component.
-host	Directs the command to a computer with a host name of dallas.
-userid	Remote user ID with which to execute the Universal Control Server process.
-pwd	Password for the user ID.

### Components

#### Universal Control

## 8.6.7 Refreshing Universal Broker via Universal Control from IBM i

This example refreshes a component on a remote system.

```
STRUCT RFSHCMPNM(*yes) HOST(dallas) USERID(joe) PWD(akksdiq)
```

Figure 8.17 Refreshing Universal Broker via Universal Control from IBM i

### Command Line Options

The command line options used in this example are:

Option	Description
RFSHCMPNM	Instruction to refresh Universal Broker on the remote system.
HOST	Directs the command to a computer with a host name of <b>dallas</b> .
USERID	Remote user ID with which to execute the Universal Control Server process.
PWD	Password for the user ID.

The REFRESH command directs Universal Broker to take the following actions:

Step	Description
Step 1	Read its configuration file <b>UNVCONF</b> and member <b>UBROKER</b> .
Step 2	Read all component definitions found in the component definition file, <b>UNVPRD420</b> / <b>UNVCOMP</b> . 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 <b>UNVCONF</b> and member <b>UACL</b> . The Broker replaces its UACL entries with the newly read entries.

### Components

#### Universal Control

## 8.6.8 Refreshing a Component via Universal Control from IBM i

This example refreshes a component on a remote system.

```
STRUCT RFSHCMPNM(*yes) RFSHCMPNM(uems) CMDID('UEM-da11as') HOST(da11as)
USERID(joe) PWD(akksdiq)
```

Figure 8.18 Refreshing Component via Universal Control from IBM i

### Command Line Options

The command line options used in this example are:

Option	Description
RFSHCMPNM	Specification for whether or not to refresh.
RFSHCMPNM	Type of component to refresh on the remote system.
CMDID	Assigns a command identifier of 'UEM-da11as' to the started component.
HOST	Directs the command to a computer with a host name of da11as.
USERID	Remote user ID with which to execute the Universal Control Server process.
PWD	Password for the user ID.

### Components

#### Universal Control

---

## 8.7 Merging Configuration Options during an Upgrade Installation Examples

This section provides examples illustrating the merging of Stonebranch Solution components' configuration options, via the Universal Products Install Merge (UPI) component, during a Stonebranch Solutions upgrade installation.

### Windows and UNIX

---

The following list provides a link to each example:

[Merge Files Using Program Defaults](#)

[Merge Files Introducing New Options](#)

[Merge Files Using Installation-Dependent Values](#)

## Files Used in Examples

The examples in this section demonstrate the expected results when Universal Install Merge is executed using two files with the contents identified in [Table 8.2](#) and [Table 8.3](#).

**Note:** Although these examples show Windows path names, the Universal Install Merge behavior demonstrated also applies to UNIX systems.

[Table 8.2](#), below, identifies the contents of `infile.txt`, a sample file in Stonebranch Solutions' 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.	

Table 8.2 Stonebranch Solutions Configuration File Sample (infile.txt)

[Table 8.3](#), below, identifies the contents of `outfile.txt`, another sample file in Stonebranch Solutions' 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

Table 8.3 Stonebranch Solutions Configuration File Sample (outfile.txt)



## 8.7.1 Merge Files Using Program Defaults

Figure 8.19, below, 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
```

Figure 8.19 Merge infile.txt into outfile.txt using program defaults

Table 8.4, below, identifies the contents of `outfile.txt` after UPIMERGE completes.

To obtain this result, UPIMERGE added options from `infile.txt` that did not exist in `outfile.txt` (that is, `installation_directory`, `message_level`, `license_key`, and so on). It also preserved the value for the `port` option by replacing the 7887 value with the currently defined 7850.

UPIMERGE also dropped the commented `host` option from `infile.txt`. UPIMERGE ignores any comments in the input file, because merging those lines into the output file would have no effect on the application's behavior.

Finally, UPIMERGE commented out the `activity_monitoring` and `event_generation` options introduced by `outfile.txt`. UPIMERGE cannot distinguish between options for new features and new values for existing options. To prevent the introduction of a new value into an application currently running with application-defined defaults, UPIMERGE's default response is to comment out any option in the output file with no match in the input file.

Keyword	Value
<code>installation_directory</code>	"C:\Program Files\Universal\UCmdMgr"
<code>message_level</code>	info
<code>port</code>	7850
<code>license_product</code>	"UNIVERSAL COMMAND MANAGER"
<code>license_customer</code>	"STONEBRANCH, INC."
<code>license_type</code>	DEMO
<code>license_expiration_date</code>	2012.12.21
<code>license_nt_servers</code>	1
<code>license_key</code>	078B-E180-64E6-3016-EA20-0CF4-58F9-B301
<code>#activity_monitoring</code>	yes
<code>#event_generation</code>	*,x100

Table 8.4 Contents of outfile.txt after default merge

## Components

### Universal Products Install Merge

## 8.7.2 Merge Files Introducing New Options

Figure 8.20, below, 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
```

Figure 8.20 Merge infile.txt into outfile.txt keeping new options

Table 8.5, below, identifies the contents of `outfile.txt` after UPIMERGE completes.

The result is almost identical to the example shown in Table 8.4. 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

Table 8.5 Contents of outfile.txt when keeping unmatched destination values

## Components

### Universal Products Install Merge

## 8.7.3 Merge Files Using Installation-Dependent Values

Figure 8.21, below, 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 Stonebranch Solutions application.

```
upimerge -dest outfile.txt -source infile.txt
-cfgtype ucmd
-installdir "D:\Program Files\Universal\UDmdMgr"
```

Figure 8.21 Merge `infile.txt` into `outfile.txt` using installation-dependent values

Table 8.6, below, identifies the contents of `outfile.txt` after UPIMERGE completes. The result is almost identical to the example shown in Table 8.4, 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 Stonebranch Solutions 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

Table 8.6 Contents of `outfile.txt` when using installation-dependent values

## Components

### Universal Products Install Merge

# Component Management

---

## 9.1 Overview

This chapter provides information on Infitran component management:

- [Component Definition](#)
- [Starting Components](#)
- [Stopping Components](#)
- [Starting / Stopping Universal Broker Examples](#)
- [Starting / Stopping Components via Universal Control Examples](#)
- [Maintaining Universal Broker Definitions in the Universal Enterprise Controller Database](#)

---

## 9.2 Component Definition

Each Infitran Server component (for Universal Data Mover, Universal Event Monitor, Universal Control, and Universal Application Container) has a Component Definition.

The Component Definition is a text file of options that defines component-specific information required by the Universal Broker.

Each Component Definition defines the following type of information:

- Component type (for Universal Event Monitor Servers only).
- Component name.
- Component command name.
- Component configuration file name.
- Component working directory path.
- Number of component instances that can run simultaneously.
- Specification for whether or not the component starts automatically when the Universal Broker starts.

The reference guide for each component contains detailed information about its Component Definition.

### 9.2.1 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 Components](#)).

For a complete explanation of the difference between demand-driven and event-driven Universal Event Monitor Servers, see [Demand-Driven vs. Event-Driven](#).

## 9.3 Starting Components

There are four ways in which Infitran components are started.

### Starting Manually

The following components are started manually and run in the background until they are stopped manually:

- Universal Broker
- Universal Enterprise Controller

The following components are started manually and run until the specified task has completed, then stop automatically:

- Stonebranch Solution utilities, such as Universal Encrypt

### Start via Manager

The following components are started on demand (that is, via their Managers) and run until the specified task has completed, then stop automatically.

- Universal Data Mover Server
- Universal Control Server
- Universal Event Monitor Server (demand-driven)

### Starting Automatically

The following components are auto-start components; that is, they start automatically when the Universal Broker starts and run until they are stopped manually:

- Universal Application Container Server
- Universal Event Monitor Server (event-driven)

**Note:** A Universal Event Monitor Server Component Definition also can specify that an event-driven server is not started automatically (see [Starting via Universal Control](#)).

### Starting via Universal Control

Universal Control can start Server components that do not require interaction with a Manager. Currently, only one Infitran component can be started via Universal Control:

- Universal Event Monitor Server (event-driven)

## 9.4 Stopping Components

Any Infitran Server component can be stopped via the Universal Control STOP command.

Authorized users also are able to use the I-Activity Monitor, a Universal Enterprise Controller client application, to stop running any Infitran Server component (if it is a component of an Agent being polled by UEC).

---

## 9.5 Starting / Stopping Universal Broker Examples

This section provides operating system-specific information for starting and stopping Universal Broker.

### z/OS

---

[Starting / Stopping Universal Broker for z/OS](#)

### Windows

---

[Starting Universal Broker for Windows](#)

### UNIX

---

[Starting Universal Broker for UNIX](#)

### IBM i

---

[Starting, Ending and Working With Universal Broker for IBM i](#)



---

## 9.5.1 Starting / Stopping Universal Broker for z/OS

---

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
```

---

## 9.5.2 Starting Universal Broker for Windows

---

Universal Broker can be executed in two different environments: Console application and 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 on the security requirements of Universal Broker and all Infitran components, see Chapter [7 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 must execute with the Local System account. The Local System account provides sufficient permissions and rights for the Broker.

The Local System account does not provide access to network resources, such as network drives or printers.

## 9.5.3 Starting Universal Broker for UNIX

Universal Broker can be executed in two different environments:

- [Daemon](#)
- [Console Application](#)

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/universa1` by default. If the PID file is in the PID directory, it is assumed that a Broker instance is executing.

### Daemon

Universal Broker can run as a UNIX daemon process. This is the preferred method of running the Broker. A daemon start-up script is provided to manage the starting and stopping of the Broker daemon. The startup script utilizes the PID file to ensure that only one instance of the Broker is executing at any one time. For this reason, the start-up script should be used to start and stop the Broker.

**Note:** Although they have the same name, the Broker daemon start-up script should not be confused with the actual Broker daemon program file.

- Startup script is installed in the primary Broker directory (that is, `./universa1/ubroker`).
- Program file is installed in the Broker's `bin` directory (that is, `./universa1/ubroker/bin`).

```
ubrokerd { start | stop | status | restart }
```

Figure 9.1 Universal Broker for UNIX - Daemon Startup Script Syntax

[Table 9.1](#), below, describes the command line arguments to the Universal Broker daemon start-up script.

Command	Description
<code>start</code>	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.
<code>stop</code>	Stops the Universal Broker daemon. If the Broker daemon is not running, the script simply returns.
<code>status</code>	Returns the status of the Universal Broker daemon, either <i>running</i> or <i>stopped</i> . If the daemon is running, the script displays its process ID.
<code>restart</code>	Performs a <b>stop</b> request followed by a <b>start</b> request.

Table 9.1 Universal Broker - Command Line Arguments to Daemon Startup Script

## 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](#) in [Console Application](#), below, for more details.)

## Console Application

The **ubroker** command starts Universal Broker as a console application.

## Console Security

Universal Broker runs with the same user ID as the user who starts it. The Broker does not require superuser rights. It only requires access to its installation directory and files, which often are created by the superuser account when the product is installed.

However, components started by the Broker also run with the same user ID as the Broker. Some of these components may require superuser rights.

See [Chapter 7 Security](#) for details on their security requirements for specific Infitran components.

## 9.5.4 Starting, Ending and Working With Universal Broker for IBM i

---

Universal Broker executes within its own IBM i subsystem, named **UNVUBR420**. The **UNVUBR420** subsystem provides a self-contained environment in which Universal Broker can be managed. The **UNVUBR420** subsystem description (object type **\*SBSD**) is named **UNVUBR420**.

The **UNVUBR420** 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 **UNVUBR420** subsystem defines one autostart entry, **UNVUBR420**. The **UBROKER** job executes with the job description **UBROKER** (object type **\*JOB**) and user profile **UNVUBR420** (object type **\*USRPRF**). Only one instance of the **UBROKER** job, which runs continuously, can be active at any one time.

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 **UNVUBR420** use the **UBROKER** job queue and class located in the product installation library. See the Stonebranch Solutions 4.2.0 Installation Guide 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 Stonebranch Solutions installation library, **UNVPRD420** (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, Stonebranch Solutions sets the job log to wrap. (See Chapter 6 IBM i Installation in the Stonebranch Solutions 4.2.0 Installation Guide for additional information.)

## Commands

The following O/S commands help manage the **UNVUBR420** subsystem.

### Start Subsystem Command (STRSBS)

Starts the Universal Broker subsystem, **UNVUBR420**.

```
STRSBS UNVPRD420/UNVUBR420
```

Figure 9.2 Universal Broker for IBM i - Subsystem Start Command

### End Subsystem Command (ENDSBS)

Ends the Universal Broker subsystem, **UNVUBR420**.

```
ENDSBS UNVUBR420
```

Figure 9.3 Universal Broker for IBM i - Subsystem End Command

### Work With Subsystem Command (WRKSBS)

Allows users to work with all active subsystems. Choose the **UNVUBR420** subsystem from the list of subsystems displayed.

```
WRKSBS
```

Figure 9.4 Universal Broker for IBM i - Subsystem Work With Command

## 9.6 Starting / Stopping Components via Universal Control Examples

This section provides examples of starting and stopping components via Universal Control.

**Note:** Currently, only Universal Event Monitor Servers can be started by Universal Control.

The examples assume that Universal Control Server is installed on a remote system named `da11as`. 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 Infitran components are provided for each example.

### z/OS

---

[Starting a z/OS Component via Universal Control](#)

[Stopping a z/OS Component via Universal Control](#)

### Windows

---

[Starting a Windows Component via Universal Control](#)

[Stopping a Windows Component via Universal Control](#)

### UNIX

---

[Starting a UNIX Component via Universal Control](#)

[Stopping a UNIX Component via Universal Control](#)

### IBM i

---

[Starting an IBM i Component via Universal Control](#)

[Stopping an IBM i Component via Universal Control](#)

## 9.6.1 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 **da11as**. 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-da11as" -host da11as -userid joe -pwd akksdiq
/*
```

Figure 9.5 Universal Control for z/OS - Start Component Example

### SYSIN Options

The SYSIN options used in this example are:

Option	Description
<b>-start</b>	Name of the component to start on the remote system.
<b>-cmdid</b>	Assigns a command identifier of " <b>UEM-da11as</b> " to the started component.
<b>-host</b>	Directs the command to a computer with a host name of <b>da11as</b> .
<b>-userid</b>	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.
<b>-pwd</b>	Password for the user ID.

### Components

#### Universal Control



## 9.6.2 Stopping a z/OS 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 **da11as**. 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 da11as -userid joe -pwd akkSdiq
/*
```

Figure 9.6 Universal Control for z/OS - Stop Example

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 **da11as** for execution.

### SYSIN Options

The SYSIN options used in this example are:

Option	Description
<b>-stop</b>	Specifies the component to stop.
<b>-host</b>	Directs the command to a computer with a host name of <b>da11as</b> .
<b>-userid</b>	Specifies the remote user ID with which to execute the stop request.
<b>-pwd</b>	Specifies the password for the user ID.

### Components

#### Universal Control

## 9.6.3 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
```

Figure 9.7 Universal Control for Windows - Start Component Example

### 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

## 9.6.4 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
```

Figure 9.8 Universal Control for Windows - Stop Component Example

### Command Line Options

The command line options used in this example are:

Option	Description
-stop	Component ID on the remote system to stop.
-host	Directs the command to a computer with a host name of <b>dallas</b> .
-userid	Remote user ID with which to execute the command. This must match the user ID originally used to start the component.
-pwd	Password for the user ID.

### Components

#### Universal Control

## 9.6.5 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
```

Figure 9.9 Start Component Example.

### 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

## 9.6.6 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
```

Figure 9.10 Universal Control Manager for UNIX - Stop Component Example 1

### Command Line Options

The command line options used in this example are:

Option	Description
-stop	Name of the component to stop.
-host	Directs the command to a computer with a host name of <b>dallas</b> .
-userid	Remote user ID with which to execute the command. This must match the user ID originally used to start the component.
-pwd	Password for the user ID.

### Components

#### Universal Control

## 9.6.7 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)
```

Figure 9.11 Start Component Example

**Note:** This example references the IBM i command by its untagged name. If you are using commands with tagged names to run Universal Control, substitute the tagged names for the untagged names.

### Command Line Options

The command line options used in this example are:

Option	Description
START	Component to start on the remote system.
CMDID	Assigns a command identifier of 'UEM-dallas' to the started component.
HOST	Directs the command to a computer with a host name of <b>dallas</b> .
USERID	Remote user ID with which to execute the Universal Control Server process. The started component, in fact, will execute with the Universal Broker's security context.
PWD	Password for the user ID.

### Components

#### Universal Control

## 9.6.8 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)
```

Figure 9.12 Universal Control for IBM i - Stop Component Example

**Note:** This example references the IBM i command by its untagged name. If you are using commands with tagged names to run Universal Control, substitute the tagged names for the untagged names.

### Command Line Options

The command line options used in this example are:

Option	Description
STOP	Component on the remote system to stop.
HOST	Directs the command to a computer with a host name of <b>dallas</b> .
USERID	Remote user ID with which to execute the stop request. This must match the user ID originally used to start the component.
PWD	Password for the user ID.

### Components

#### Universal Control

---

## 9.7 Maintaining Universal Broker Definitions in the Universal Enterprise Controller Database

This section contains examples demonstrating the use of the UECLoad utility.

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

---

### z/OS and Windows

---

[List All Defined Brokers](#)

[Export a Specific Defined Broker](#)

[Export Events](#)

[Retrieve Archived File and Export](#)

[Delete a Specific Defined Broker](#)

[Add Specific Defined Broker via deffile](#)

[Add Existing Brokers to a Broker Group](#)

[Delete Existing Brokers to a Broker Group](#)

---

### z/OS

---

[Export Events into ARC Format for z/OS](#)

[Retrieve Archived File and Export into XML for z/OS](#)

---

### Windows

---

[Export Events into ARC Format for Windows](#)

[Retrieve Archive File and Export into CSV for Windows](#)

**Note:** All examples in this section are implemented with use of the [UECLoad Utility](#) component.



---

## 9.7.1 List All Defined Brokers

---

Figure 9.13, below, illustrates the output of a user-friendly format of the Brokers defined in the UEC database.

```
ueclload -port 8778 -userid joe -pwd akkSdiq -list -broker_name "**"
```

Figure 9.13 UECLoad - List All Defined Brokers

---

## 9.7.2 Export a Specific Defined Broker

---

Figure 9.14, below, illustrates the output of a Broker defined in the UEC database in a format suitable for use within a broker definition file.

Note: Although this command is illustrated on two lines, it should be entered as one line at the command prompt.

```
ueclload -port 8778 -userid joe -pwd akkSdiq -level audit  
-export -broker_name mybroker1
```

Figure 9.14 UECLoad - Export a Specific Defined Broker

---

## 9.7.3 Export Events

---

Figure 9.15, below, illustrates the export of an events file into CSV format.

Note: Although this command is illustrated on two lines, it should be entered as one line at the command prompt.

```
ueclload -port 8778 -userid joe -pwd akkSdiq -level audit -export EVENTS  
-stime *-5 -etime * -format CSV -deffile events.csv
```

Figure 9.15 UECLoad - Export Events

## 9.7.4 Retrieve Archived File and Export

---

Figure 9.16, below, illustrates the retrieval of an archived events file and its export into CSV format.

Note: Although this command is illustrated on two lines, it should be entered as one line at the command prompt.

```
ueclload -arcfile c:\test.arc -export EVENTS -stime 2006/10/07 -etime  
2008/01/01 -level audit -format CSV -deffile c:\test.csv
```

Figure 9.16 UECLoad - Retrieve Archived File and Export

## 9.7.5 Delete a Specific Defined Broker

---

Figure 9.17, below, illustrates the deletion of a Broker defined in the UEC database. Specifically, Broker `mybroker1` is deleted from use of UEC.

Note: Although this command is illustrated on two lines, it should be entered as one line at the command prompt.

```
ueclload -port 8778 -userid joe -pwd akkSdiq -level audit  
-delete -broker_name mybroker1
```

Figure 9.17 UECLoad - Delete a Specific Defined Broker

## 9.7.6 Add Specific Defined Broker via deffile

Figure 9.18, below, illustrates the addition of a group of Broker definitions specified within a definition file in the UEC database. The name `sample_deffile` represents the name of the created file.

Note: Although this command is illustrated on two lines, it should be entered as one line at the command prompt.

```
ueclload -port 8778 -userid joe -pwd akkSdiq -level audit
        -add -deffile sample_deffile
```

Figure 9.18 UECLoad - Add Specific Defined Broker via a Definition File

Figure 9.19, below, 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>
```

Figure 9.19 UECLoad - Definition File used for Adding Specific Defined Broker

---

## 9.7.7 Add Existing Brokers to a Broker Group

---

Figure 9.20, below, illustrates the addition of existing Universal Brokers to a Broker group.

Note: Although this command is illustrated on two lines, it should be entered as one line at the command prompt.

```
ueclload -port 8778 -userid joe -pwd akkSdiq -add -deffile brokers  
-groups "Test 1, Test 2, Test 3"
```

Figure 9.20 UECLoad - Add Existing Brokers to a Broker Group

---

## 9.7.8 Delete Existing Brokers to a Broker Group

---

Figure 9.21, below, illustrates the deletion of existing Universal Brokers from a Broker group.

Note: Although this command is illustrated on two lines, it should be entered as one line at the command prompt.

```
ueclload -port 8778 -userid joe -pwd akkSdiq -delete -deffile brokers  
-groups "Test 2, Test 3"
```

Figure 9.21 UECLoad - Delete Existing Brokers to a Broker Group

## 9.7.9 Export Events into ARC Format for z/OS

Figure 9.22, below, 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
```

Figure 9.22 UECLoad for z/OS - Export Events into ARC Format

## 9.7.10 Retrieve Archived File and Export into XML for z/OS

Figure 9.23, below, 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
```

Figure 9.23 UECLoad for z/OS- Retrieve Archived File and Export into XML

## 9.7.11 Export Events into ARC Format for Windows

---

Figure 9.24, below, illustrates the export of events into an ARC format file on Windows.

```
ueclload -export EVENTS -userid admin -pwd admin -format ARC -stime 2008/07/24  
-etime 2008/07/24 -deffile c:\test.xml -arcfile c:\test.arc
```

Figure 9.24 UECLoad for Windows - Export Events into ARC Format

## 9.7.12 Retrieve Archive File and Export into CSV for Windows

---

Figure 9.25, below, illustrates the retrieval of an archived file and its export into CSV on Windows.

```
ueclload -arcfile c:\test.arc -export EVENTS -stime 2006/10/07 -etime  
2008/01/01 -level audit -format CSV -deffile c:\test.csv
```

Figure 9.25 UECLoad for Windows - Retrieve Archived File and Export into CSV

**Note:** `-port`, `-userid`, and `-pwd` are not used, since no connection is made to UEC for this operation.

# Messaging and Auditing

---

## 10.1 Overview

All Stonebranch Solutions 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.

This section describes the message and audit facilities that are common to all Stonebranch Solutions components. (See the individual Stonebranch Solutions 4.2.0 documentation for detailed technical information.)

## 10.2 Messaging

This section describes the Stonebranch Solutions messaging facility:

[Message Types](#)

[Message ID](#)

[Message Levels](#)

[Message Destinations](#)

### 10.2.1 Message Types

---

There are six types (or severity levels) of Stonebranch Solutions messages. (The severity level is based on the type of information provided by those messages.)

1. Audit messages 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.
2. Informational messages document the actions being taken by a program. They help determine the current stage of processing for a program. Informational messages also document statistics about data processed.
3. Warning messages document unexpected behavior that may cause or indicate a problem.
4. Error messages document program errors. They provide diagnostic data to help identify the cause of the problem.
5. Diagnostic messages document diagnostic information for problem resolution.
6. Alert messages 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 Stonebranch Solutions component lets you specify which messages are written (see Section [10.2.3 Message Levels](#)).



---

## 10.2.2 Message ID

---

Each message is prefixed with a message ID that identifies the message.

The message ID format is **UNVnnnn1**, where:

- **nnnn** is the message number.
- **1** is the message severity level:
  - **A** (Audit)
  - **I** (Informational)
  - **W** (Warning)
  - **E** (Error)
  - **T** (alerT)
  - **D** (Diagnostic)

Note: The Stonebranch Solutions 4.2.0 Messages and Codes document identifies all messages numerically, by product, using the **nnnn** message number.

---

## 10.2.3 Message Levels

---

Each Stonebranch Solutions component includes a `MESSAGE_LEVEL` configuration option that lets you select which levels (that is, severity levels) of messages are to be written.

- *Audit* specifies that all audit, informational, warning, and error messages are to be written.
- *Informational* specifies that all informational, warning, and error messages are to be written.
- *Warning* specifies that all warning and error messages are to be written.
- *Error* specifies that all error messages are to be written.
- *Trace* specifies that a trace file is created, to which data used for program analysis will be written. The trace file name and location are component-dependent (see the appropriate Stonebranch Solutions 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.

## 10.2.4 Message Destinations

The location to which messages are written is the message destination.

Some Stonebranch Solutions 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

Stonebranch Solutions 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.

### 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 <code>/var/opt/universal/log</code> . This can be changed with the <code>LOG_DIRECTORY</code> option. All messages written to log files include a date and time stamp and the program's process ID.
SYSTEM	Messages are written to the <code>syslog</code> daemon. Not all programs provide this destination. Universal programs that execute as daemons write to the <code>syslog</code> 's daemon facility. All messages include the programs process ID. If an error occurs writing to the <code>syslog</code> , the message is written to the system console.

## 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.

## 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.

## 10.3 Auditing

Within Stonebranch Solutions, 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 Stonebranch Solutions component, a user authentication failure, or a file transfers completing.

The Universal Event Subsystem (UES) provides the means by which Stonebranch Solutions 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 Stonebranch Solutions components.

Stonebranch Solutions consist of a set of components distributed across a computer network. The components work together to perform some unit of work. The components that are working together have an association that must be maintained in the event data. For that reason, UES event records not only include information about the event, but also information about associations between the components reporting the events.

The Universal Enterprise Controller (UEC) maintains a central UES database for all event data within its domain of responsibility. The UES database contains all UES event records collected by UEC from Universal Broker components that are defined to it. The UES database provides medium-term persistent storage for the UES events. Periodically, the UES database events must be exported to long-term storage in order to maintain a historical record of events. If the export is not performed periodically, the UES database will continue to grow and eventually exhaust all disk space available to it.

Examples of components and their associations are:

- Universal Command Manager is associated with a remote Universal Command Server, and the Universal Command Server is associated with the job process it has started on behalf of the Universal Command Manager.
- Universal Data Mover Manager is associated with a remote Universal Data Mover Server, and the Universal Data Mover Server is associated with a file being transferred on behalf of the Universal Data Mover Manager.

The components and their associations partly define the Stonebranch Solutions architecture. This section provides the necessary understanding of the Stonebranch Solutions architecture as presented by the UES event data.

---

## 10.4 Creating Write-to-Operator Messages Examples

This section provides examples demonstrating how to create write-to-operator message in a z/OS USS environment via the Universal Write-to-Operator utility.

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

### z/OS

---

[USS UWTO for z/OS Console](#)

[USS UWTO for z/OS Console and Wait for Reply](#)

## 10.4.1 USS UWTO for z/OS Console

Figure 10.1, below, 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"
```

Figure 10.1 Universal WTO - Issue WTO to z/OS 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	Specifies the 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

## 10.4.2 USS UWTO for z/OS Console and Wait for Reply

Figure 10.2, 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
```

Figure 10.2 Universal WTO - Issue WTOR to z/OS Console

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

---

## 11.1 Overview

Infitran component error messages are translated, by the Universal Message Translator utility, into return (exit) codes based on a user-defined translation table.

Every command ends with a return code that indicates the success or failure of the command execution. Typically, a return code of 0 indicates success; all other codes indicate failure.

However, a small number of commands do not set their return code under failure conditions; instead, they issue error messages. Based on the user-defined translation table, Universal Message Translator translates these error messages into return codes.



## 11.2 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.

### 11.2.1 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

The translation table entry fields are:

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.

Table 11.1 Universal Message Translator – Translation Table

### 11.2.2 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 [Table 11.2](#), below.

Return Code	Message ID	Message Severity
1 – 10	UNV0344	10
11 – 20	UNV0345	20
21 – 30	UNV0346	30
31 and higher	UNV0347	40

Table 11.2 Universal Message Translator for IBM i - Return Codes

---

## 11.3 Message Translation Examples

### All Operating Systems

---

[Universal Message Translator \(Part 1\)](#)

[Universal Message Translator \(Part 2\)](#)

### z/OS

---

[Execute Universal Message Translator from z/OS](#)

### Windows

---

[Execute Universal Message Translator from Windows](#)

### UNIX

---

[Execute Universal Message Translator from UNIX](#)

### IBM i

---

[Execute Universal Message Translator from IBM i](#)

**Note:** IBM i examples reference the IBM i commands by their untagged names. If you are using commands with tagged names to run Universal Message Translator, substitute the tagged names for the untagged names.

## 11.3.1 Universal Message Translator (Part 1)

**Note:** This example is not specific to a particular operating system.

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.
```

Figure 11.1 Universal Message Translator - Example 1, Message File

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.

```
# UMET Translation Table 1
#
# Message Mask                Exit Code
# -----
# "*error*"                   8
```

Figure 11.2 Universal Message Translator - Example 1, Translation Table 1

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.

```
# UMET Translation Table 2
#
# Message Mask                Exit Code
# -----
# "Ending due to error."      8
```

Figure 11.3 Universal Message Translator - Example 1, Translation Table 2

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.

## Components

### Universal Message Translator

## 11.3.2 Universal Message Translator (Part 2)

*This example continues from [Universal Message Translator \(Part 1\)](#).*

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
```

Figure 11.4 Universal Message Translator - Example 2, Message File

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`.

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
```

Figure 11.5 Universal Message Translator - Example 2, Translation Table 1

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](#)

### 11.3.3 Execute Universal Message Translator from z/OS

Figure 11.6, below, 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.
/*
```

Figure 11.6 Universal Message Translator - Execute from z/OS

The parameter **-TABLE** 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** 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
<b>-table</b>	Translation table file name.
<b>-level</b>	Level of messages that will be displayed.

#### Components

##### Universal Message Translator

## 11.3.4 Execute Universal Message Translator from Windows

Figure 11.7, below, illustrates the execution of Universal Message Translator from Windows.

```
umet -table c:\umettable.txt -file c:\umetfile.txt -level verbose
```

Figure 11.7 Universal Message Translator - Execute from Windows

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
<b>-table</b>	Translation table file name.
<b>-level</b>	Level of messages that will be displayed.
<b>-file</b>	Input message file name. If the option is not specified, UMET reads its input from <b>stdin</b> .

### Components

#### Universal Message Translator

## 11.3.5 Execute Universal Message Translator from UNIX

Figure 11.8, below, illustrates the execution of Universal Message Translator from UNIX.

Although the command is shown on two lines, it should be entered on one line at the command prompt or within a script, or it can be continued within the script with the UNIX continuation character `\`.

```
/opt/universal/ucmdsrv-2.2.0/bin/umet -table /tmp/umettable.txt  
-file /tmp/umetfile.txt -level verbose
```

Figure 11.8 Universal Message Translator - Execute from UNIX

The parameter `-t` points to the file, which defines the return codes with which to end this process, based on matching text. The first column of the file defines the text to match; the second defines the return code to set if the matching text exists in the file defined by `-f`. All messages will be written to **stdout**.

### Command Line Options

The command line options used in this example are:

Option	Description
<code>-table</code>	Translation table file name.
<code>-level</code>	Level of messages that will be displayed.
<code>-file</code>	Input message file name. If the option is not specified, UMET reads its input from <b>stdin</b> .

### Components

#### Universal Message Translator



## 11.3.6 Execute Universal Message Translator from IBM i

Figure 11.9, below, illustrates the execution of Universal Message Translator from IBM i.

```
STRUME MSGFILE(input_file) MSGMBR(member) TBL(table_file) TBLMBR(member)
MSGLEVEL(*VERBOSE)
```

Figure 11.9 Universal Message Translator - Execute from IBM i

The parameter **TBL/TBLMBR** 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 **MSGFILE/MSGMBR**.

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 <b>stdin</b> , which is allocated to the terminal for interactive jobs and to QINLINE for non-interactive jobs.

### Components

#### Universal Message Translator

# Monitoring and Alerting

---

## 12.1 Overview

The Monitoring and Alerting feature of Infitran provides for the monitoring the status and activity of all Infitran Agents in an enterprise and the posting of alerts regarding the statuses.

Monitoring is provided through continuous [Monitoring of All Agents](#) or by [Querying for Job Status and Activity](#) of a specific Agent.

---

## 12.2 Monitoring of All Agents

Infitran provides for the continuous monitoring of all Agents in an enterprise through its Universal Enterprise Controller component.

### 12.2.1 Monitored Information

---

Infitran monitors for four 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. Files (active, completed, and failed) transferred by Infitran for all Agents being monitored.
4. Systems (Agents and SAP systems) being monitored.

This information can be viewed via the I-Activity Monitor client application.

### 12.2.2 Polling

---

Infitran periodically polls each Agent and SAP system in an enterprise in order to retrieve its status information.

It determines whether or not a change in status of the Agent or SAP system has occurred since the last poll. If the status has changed, it sends this information to the I-Activity Monitor.

### 12.2.3 Alerts

---

Infitran sends out alerts to any connected Agent-monitoring applications whenever:

- Agent is unreachable.
- Agent is not responding.
- Agent component enters an orphaned or disconnected state.

These alerts are posted to the:

- Event Log (when running under Windows)
- Console (when running under z/OS)

Automation tools can be used in conjunction with these messages to perform operations based on agent failures.

---

## Alert Types

UEC creates three types of alerts:

- **Agent Down**  
UEC was unable to establish a connection with the broker on the last poll attempt.
- **Component Disconnected**  
Server is not connected to the Manager. This occurs when a network error has occurred, the manager halted, or the manager host halted. The server is executing with either the network fault tolerant protocol, is restartable, or both.  
  
Note: The Server cannot determine whether or not the Manager is still executing because it cannot communicate with it.
- **Component Orphaned**  
Manager has terminated. The manager sends a termination message to the server to notify it of its termination prior to terminating. This state only occurs if the server is restartable.

## 12.3 Querying for Job Status and Activity

Infitran has the ability to query any specific Universal Broker in an enterprise for Broker-related, and active component-related, activity via the Universal Query utility.

Universal Query returns information for a Universal Broker that is installed on the host, as specified by configuration options on the command line or in a configuration file. Information regarding the components managed by a particular 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.

---

## 12.4 Querying for Job Status and Activity Examples

This chapter provides user scenarios for the Querying Job Status feature of Infitran.

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

---

### All Operating Systems

---

[Universal Query Output](#)

---

### z/OS

---

[Universal Query for z/OS](#)

---

### UNIX and Windows

---

[Universal Query for UNIX and Windows](#)

---

### IBM i

---

[Universal Query for IBM i](#)

## 12.4.1 Universal Query Output

Figure 12.1, below, illustrates an example of the output generated by the execution of the Universal Query command.

This sample output is from the execution of Universal Query to host `dallas.domain.com` using a NORMAL report.

```

                                Universal Query Report
                                for
                                Mon 20 May 2010 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.....: 4.2.0 Level 0 Release Build 108
    Ubroker Service.....: UNKNOWN
    Ubroker Status.....: Active

Component ID.....: 1121367481
Component Name.....: ucmd
Component Description.....: Universal Command Server
Component Version.....: 4.2.0 Level 0 Release Build 108
Component Type.....: ucmd
Component Process ID.....: 773
Component Start Time.....: 05:53:39 PM
Component Start Date.....: 05/20/2010
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.: 07/20/2010
Component MGR Restartable.: NO
Component Comment.....: sleep for 60 secs on dallas
```

Figure 12.1 Universal Query Output

## 12.4.2 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
/*
```

Figure 12.2 Universal Query for z/OS - Listing Active Components

All active component information for server dallas will be printed to DD statement **SYSOUT**.

### SYSIN Option

The SYSIN option used in this example is:

Option	Description
-host	Directs the command to a computer with a host name of <b>dallas</b> .

### Components

#### Universal Query



## 12.4.3 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
```

Figure 12.3 Universal Query for UNIX and Windows - Listing Active Components

All active component information for the `localhost` server will be printed to stdout.

### Command Line Option

The command line option used in this example is:

Option	Description
<code>-host</code>	Directs the command to the <code>localhost</code> .

### Components

#### Universal Query

## 12.4.4 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)
```

Figure 12.4 Universal Query for IBM i (specific port) - Listing Active Components

This command provides active component information for the `localhost` server listening on port 4990 will be printed to stdout.

```
STRUQR HOST(fortworth)
```

Figure 12.5 Universal Query for IBM i (default port) - Listing Active Components

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 <code>localhost</code> .
PORT	TCP port on the remote server.

### Components

#### Universal Query

# Windows Event Log Dump

---

## 13.1 Overview

Infitran provides the ability to select records from a Windows event log and write them to a specified output file via its Universal Event Log Dump utility.

All records from a log can be dumped, or event records can be selected according to the date and time that they were generated.

Universal Event Log Dump can be run any time as a stand-alone application. It also is designed to work with 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 (`ue1d`) followed by a list of configuration options.

---

## 13.2 Windows Event Log Dump Examples

### Windows

---

[Execute Universal Event Log Dump from z/OS Manager](#)

[Execute Universal Event Log Dump from a Windows Server](#)

## 13.2.1 Execute Universal Event Log Dump from z/OS Manager

Figure 13.1, below, 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=h1q.userid(userid)
//SCRIPTDD DD *
ue1d -type APPLICATION -stime '*-1,15:00 PM'
//SYSIN DD *
-script SCRIPTDD
-encryptedfile LOGONDD
-host dallas
/*
```

Figure 13.1 Universal Event Log Dump - Execution from z/OS Manager

The JCL procedure **UCMDPRC** is used to execute the **ue1d** 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**.

### Command Line Options

The command line options used in this example are:

Command Options	Description
<b>-type</b>	Event log to be dumped.
<b>-stime</b>	Starting date and time.

## SYSIN Options

The SYSIN options used in this example are:

Option	Description
<code>-script</code>	ddname from which to read a script file. The script file is sent to the remote system by UCMD Manager for execution
<code>-encryptedfile</code>	File from which to read an encrypted command options file.
<code>-host</code>	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](#)

## 13.2.2 Execute Universal Event Log Dump from a Windows Server

Figure 13.2, below, 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.

```
ue1d -type APPLICATION -stime "*-1,15:00 PM" -file c:\application.log
```

Figure 13.2 Universal Event Log Dump - Execution from Windows Server

### Command Line Options

The command line options used in this example are:

Command Options	Description
-type	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

# Databases

---

## 14.1 Overview

Some Infitran components provide features that rely upon a set of databases for their implementation. Such features include fault tolerance, managed configuration, event subsystem (UES) data collection, and event monitoring.

Unless otherwise noted, the Universal Broker owns all databases and performs all direct database access. Universal Broker processes and responds to all database access requests it receives from individual Infitran components.



## 14.2 Component Information Database

The component information database records information about all Infitran server components that the Universal Broker manages. It is opened during Universal Broker start-up processing.

The information captured by the Universal Broker includes, but is not limited to, the component's process ID, start time, current state, and end time.

One important aspect of this database is its ability to record the current state of an Infitran server component. Each time a component's state changes, it sends a notification to Universal Broker, which updates that component's record. For the Infitran components that offer it, this component state provides the basis for reconnect functionality, otherwise known as network fault tolerance (see Chapter [15 Fault Tolerance Implementation](#)).

When an Infitran server process finishes executing and its component state indicates that it has completed, Universal Broker deletes that component's information from the database.

The Universal Broker stores Infitran component information in the `bcomponent.db` and `scomponent.db` database files.

### Windows

The database file default location is:

- `C:\Program Files\Universal\spool\ubroker` (32-bit Windows systems)
- `C:\Program Files (x86)\Universal\spool\ubroker` (64-bit Windows systems)

### UNIX

The database file resides in the `/var/opt/universal/spool` directory.

### z/OS

Infitran components access this file via an HFS- or ZFS-allocated dataset, which is mounted on the z/OS UNIX System Services (USS) file system. Universal Broker is capable of dynamically mounting this database during start-up, if it is not already mounted.

### IBM i

The database, `UBR_CMP_DB`, is located in the spool library, `UNVSPL420`.

## 14.3 Universal Event Monitor Databases

To support the Universal Event Monitor (UEM) component, the Universal Broker provides and manages the following databases:

- [Event Definition Database](#)
- [Event Handler Database](#)
- [Event Spool Database](#)

The database files are local to each system. The Stonebranch Solutions install script is responsible for creating the database directory. If the Universal Broker attempts to open a database file that does not exist, it will create that database.

### UNIX

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

### Windows

The default UEM database directory is:

- `C:\Program Files\Universal\spool\ubroker` (32-bit Windows systems).
- `C:\Program Files (x86)\Universal\spool\ubroker` (64-bit Windows implementations).

For additional information that applies to all database files, including restrictions on location and space requirements, see the Stonebranch Solutions 4.2.0 Installation Guide.

**Note:** UEM Server is only available for UNIX and Windows. The UEM databases are used only on those operating systems.

---

## 14.3.1 Event Definition Database

---

The Universal Event Monitor (UEM) Server stores information about the events that it monitors in the event definition database.

An event definition record describes a system event and provides the information that UEM uses to track an event occurrence and test for its completion. An event definition record also may contain information that UEM uses to respond to (that is, "handle") an event's successful completion, its failure to complete, and even its failure to occur.


An event-driven UEM Server relies upon stored event definitions for its input. When an event-driven UEM Server starts, it asks the Broker for all event definitions assigned to it. If no event definitions are assigned to a particular event-driven Server, that Server continues to execute but will not do any actual event monitoring.

A demand-driven UEM Server also may obtain its input from a stored event definition record, but it is not required. Typically, a demand-driven Server receives its input from the UEM Manager's command line parameters.

Event definition records are added and maintained with the [UEMLoad Utility](#).

## 14.3.2 Event Handler Database

An event handler record describes the action that Universal Event Monitor (UEM) should take in response to a monitored event's outcome. This action, or response, is simply a system command or script that UEM executes upon an event's completion, failure to complete, or failure to occur. The UEM Server executes these processes in a secure context, using user account credentials stored in the event handler record.



Security is a primary concern within all Stonebranch Solutions.

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

**Stoneman's Tip**

An event-driven UEM Server relies upon stored event handlers to determine its response to the events that it monitors. The event definition records that describe events to UEM also contain the IDs of event handler records that UEM should use to respond to those events.

A demand-driven UEM Server also may respond to an event using a stored event handler record, but it is not required. Typically, a demand-driven Server relies upon the UEM Manager's command line parameters to describe the actions that it should take in response to the event that it monitors.

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

Event handler records are added and maintained with the [UEMLoad Utility](#).

### 14.3.3 Event Spool Database


Universal Event Monitor (UEM) records its monitoring activity in the event spool database.

It is possible for UEM to detect multiple occurrences of any single event that it monitors. UEM creates a record in the spool database for each event occurrence that it detects and tracks. UEM maintains the current state of an event occurrence from initial detection through the completion of any event handlers.

If an event definition goes inactive before UEM detects any occurrences of that event, UEM creates a single spool entry to record the expired event.

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

Typically, any spool records created for an event are deleted when the Broker detects the completion of the UEM Server that monitored the event. However, when an event-driven UEM Server completes, any records that indicate work in progress (for example, tracking of an event occurrence, execution of an event handler) are retained for possible recovery when the event-driven Server is restarted. For additional information on recovery of event spool records, see Chapter 7 [Universal Event Monitor Server](#) in the [Universal Event Monitor 4.2.0 Reference Guide](#).

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

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

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

Currently, event spool records can only be viewed with the Universal Spool List utility (`uslist`). Information on using Universal Spool List to view event spool records can be found in Chapter 7 [Universal Event Monitor Server](#) in the [Universal Event Monitor Reference Guide](#). For information on all Stonebranch Solutions utilities, see the [Stonebranch Solutions Utilities Reference Guide](#).

---

## 14.3.4 Controlling UEM Database Access

---

Universal Broker is responsible primarily for providing access to the Stonebranch Solutions databases. However, there are utilities provided, including the Universal Spool List (`us1ist`) and Universal Spool Remove (`us1rm`), that can be used to access the databases directly. While these utilities should be used only following a recommendation from and with the assistance of Stonebranch, Inc. [Customer Support](#), they are documented fully in the [Stonebranch Solutions Utilities 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 has access to them.

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

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

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

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

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

Section [7.5 Universal Access Control List](#) provides a more thorough overview of the UACL feature. For information on the specific UACL used to control access to the event definition and event handler databases, see the [DATABASE\\_MAINTENANCE\\_ACL](#) UACL entry in the [Universal Event Monitor Reference Guide](#).

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

---

## 14.4 Universal Enterprise Controller Databases

Universal Enterprise Controller (UEC) uses databases to maintain agent, user, configuration, and event data.

### 14.4.1 Database Files

---

The UEC databases reside in three files:

1. **uec.db** contains the definitions of agents, groups, users, SAP systems, and a record of updates to distributed components' configurations in a managed environment.
2. **uec\_evm.db** contains the Universal Event Subsystem (UES) persistent events.
3. **uec\_tmp.db** contains UES events and component information that is temporary to support I-Activity Monitor. This file is deleted and created upon restart of UEC.

### 14.4.2 Database Management

---

#### Automated Database Cleanup

Two routines are run to clean up records that meet their expiration criteria from their UEC database.

1. Routine for monitor events used for I-Activity Monitor.
2. Routine for persistent events stored for the Universal Event Subsystem.

Both routines execute at UEC start-up. Thereafter, they are scheduled to execute one hour after the previous execution's completion. At the time of execution, all records that meet the expiration criteria are removed from their UEC database.

The following UEC configuration options control database record retention:

- **COMMIT\_COMPLETE\_EXPIRATION**
- **COMMIT\_INCOMPLETE\_EXPIRATION**
- **MONITOR\_EVENT\_EXPIRATION**
- **PERSISTENT\_EVENT\_EXPIRATION**

## Memory Management

Berkeley DB uses a temporary cache in memory to manage its databases. If this cache becomes sufficiently large, it must be written to disk.

Berkeley DB has a default location for storing temporary cache files, but if UEC cannot access that location, or there is no space to write these files in the default location, the following error can occur in UEC, and UEC shuts down:

```
UNV4301D Database error: 'temporary: write failed for page xxxxx'
```

To work around this issue, the following steps will write the temporary cache files to the UEC database directory:

1. For z/OS installations, mount the **UECDB** HFS or zFS dataset.
2. Inside the UEC database directory (or, on z/OS, the mount point), create a text file named **DB\_CONFIG**.
3. Inside the **DB\_CONFIG** file, add the following string:  

```
set_tmp_dir *dbpath*
```

Where **dbpath** is the path to the location in which the database files reside.
4. Start / restart UEC.



## 14.5 Database Backup and Recovery

Stonebranch Solutions databases, on operating system's other than IBM i, are implemented using Oracle's Berkeley Database product.

Recovering from database corruption requires the following steps:

1. Dump the corrupted database to a file using the Stonebranch Solutions [Universal Database Dump](#) utility.
2. Reload the database from the dump file using the Stonebranch Solutions [Universal Database Load](#) utility.

Database corruption can occur if the system or address space that is managing the databases ends abnormally. A Stonebranch Solutions program that utilizes databases should not be terminated abnormally.

Abnormal methods of termination include:

- z/OS CANCEL or FORCE command.
- UNIX SIGKILL signal.
- Windows process termination through the Task Manager.

### 14.5.1 Database Backups

---

Database recovery is not a replacement for database backups. If the data maintained by the product in the database has long term value, the databases must be periodically backed up.

---

## 14.5.2 General Database Recovery Procedures

---

Generally speaking, database recovery follows the same steps independent of platform and database file.

Multiple attempts may be necessary in order to successfully recover from database corruption. Stonebranch Inc. recommends that you begin with the least aggressive recovery method and only proceed to more aggressive methods if necessary.

For the first recovery attempt, execute the [Universal Database Dump](#) utility with the `-r` (lowercase) command line option. This option instructs the utility to recover as much data as possible. Depending on the extent of database corruption, this may result in a recovered database with some incomplete key/data pairs.

Reload the database using the [Universal Database Load](#) utility, and specify the `-o` option. This option instructs the utility to remove the underlying database file, which results in a clean reload from the dump file.

If the database passes validation when you restart the application, it is likely that all data was successfully recovered and no additional recovery attempts are necessary.

If the database fails validation, rerun the Universal Database Dump utility and omit the `-r` option. This results in a dump of only the most complete data. While this improves the chances for successful recovery, some data loss is likely. Rerun the Universal Database Load utility and restart the application.

If both recovery attempts fail, you may delete the corrupted database and restart the application. This results in a total loss of data, but will allow the application to execute. The application will create the missing database during startup.

The following sections describe platform- and database-specific recovery procedures.

## 14.5.3 Database Recovery for Universal Broker

---

Universal Broker uses databases to maintain component information, configuration information, and event data. A corrupted database will prevent the Broker from executing.

Database recovery procedures depend partly on the operating system on which the Broker is executing. The following sections describe the procedures for each operating system.

### z/OS

The Universal Broker started task must be down to perform database recovery. A backup of either the database file being recovered or the entire HFS or zFS data set should be created before recovery is attempted.

A sample database recovery job is provided in member **UBRDBREC** in the **SUNVSAMP** library. The job uses the Universal Database Utilities to dump and reload a database file.

All databases are located in the HFS or zFS product data set **#HLQ.UNV.UNVDB**. The HFS or zFS data set must be mounted prior to running **UBRDBREC**. Refer to the Stonebranch Solutions 4.2.0 Installation Guide for information on mounting the HFS or zFS data set, if necessary.

The user ID with which the recovery job runs requires appropriate permissions to the root directory of the HFS or zFS data set and to the database file. Write access is required to the directory and read and write access is required to the database file.

Customize **UBRDBREC** to meet local JCL and installation requirements. Specify the database file name to recover on the **PARM** keyword of the **EXEC** statement of both steps (the dump and load steps). When all modifications are complete, submit the job. All steps should end with return code 0.

### UNIX

The Broker daemon must be down to perform database recovery. A backup of either the database file being recovered or the entire directory should be created before recovery is attempted.

A sample database recovery script is provided in file **ubrdbrrec** in the **/opt/universa1/ubroker/bin** directory. The script uses the Universal Database Utilities to dump and reload a database file.

The default location of all Universal Broker databases is the **/var/opt/universa1/spool** directory.

The user ID with which the recovery script runs requires appropriate permissions to the database directory and to the database file. Write access is required to the directory and read and write access is required to the database file.

The **ubrdbrec** script accepts an optional argument: the database file name to recover. If no database file name is specified, the **ues.db** database is recovered. The script ends with exit code 0 if successful and a non-zero exit code if it failed.

## Windows

The Broker service must be stopped to perform database recovery. A backup of either the database file being recovered or the entire directory should be created before recovery is attempted.

A sample database recovery batch file is provided in file **ubrdbrec.bat** in the "**Program Files\Universal\UBroker\bin**" directory. The batch file uses the Universal Database Utilities to dump and reload a database file.

The default location of all Universal Broker databases is directory "**Program Files\Universal\spool\ubroker**".

The user ID with which the recovery script runs requires appropriate permissions to the database directory and to the database file. Write access is required to the directory and read and write access is required to the database file.

The **ubrdbrec.bat** batch file accepts an optional argument: the database file name to recover. If no database file name is specified, the **ues.db** database is recovered. The batch file ends with exit code 0 if successful and a non-zero exit code if it failed.

## IBM i

The Universal Broker subsystem, **UNVUBR420** (by default), must be down in order to perform database recovery. Use standard IBM i database recovery procedures and attempt to restart the Universal Broker subsystem.

If the problem persists, restore the failing database file. The entire Universal Spool file library may be required if restoring individual files fails to correct the problem. As a last resort, delete all files in the Universal Spool file library and restart **UNVUBR420**.

Deleting the files from the Universal Spool library will result in loss of all data stored in those files, including spooled output for Manager Fault Tolerant jobs. All affected jobs may need to be re-run.

---

## 14.5.4 Database Recovery for Universal Enterprise Controller

---

If Universal Enterprise Controller (UEC) terminates abnormally, it creates the file `uec.hf` in the database directory, which prompts UEC to initiate database verification upon restart.

Upon start-up, if UEC determines that an abnormal termination occurred, a verification process is performed on the database files. Verification tests the integrity of the files and determines if they are suitable for opening. If errors are detected and the integrity of the file is compromised, UEC reports the errors to the console and UEC immediately shuts down.

The Universal Database Dump (**UDBDUMP**) utility and the Universal Database Load (**UDBLOAD**) utility enable recovery from a corrupted Berkeley database. (For detailed information on these utilities, see the [Stonebranch Solutions Utilities Reference Guide](#).)

Database recovery procedures depend partly on the operating system on which UEC is executing: z/OS or Windows. The following sections describe the procedures for each operating system.

### z/OS

The UEC started task must be down to perform database recovery. A backup of either the database file being recovered or the entire HFS or zFS data set should be created before recovery is attempted.

A sample database recovery job is provided in member **UECDBREC** in the **SUNVSAMP** library. The job uses the Universal Database Utilities to dump and reload a database file.

All databases are located in the HFS or zFS product data set **#HLQ.UNV.UECDB**. The HFS data set is allocated to the **UNVDB** ddname in both the dump and load steps. The HFS or zFS data set must be mounted prior to running **UECDBREC**. See the Stonebranch Solutions 4.2.0 Installation Guide for additional information on mounting the HFS or zFS data set.

The user ID with which the recovery job runs requires appropriate permissions to the root directory of the HFS data set and to the database file. Write access is required to the directory and read and write access is required to the database file.

Customize **UECDBREC** to meet local JCL and installation requirements. All UEC databases are recovered by the job. When all modifications are complete, submit the job. All steps should end with return code 0.

## Windows

The UEC service must be stopped to perform database recovery. A backup of either the database file being recovered or the entire directory should be created before recovery is attempted.

A sample database recovery batch file is provided in file `uecdbrec.bat` in the "**\Program Files\Universal\UECtlr\bin**" directory. The batch file uses the Universal Database Utilities to dump and reload a database file.

The default location of all UEC databases is "**\Program Files\Universal\UECtlr**".

**Note:** Stonebranch has identified an issue with upgrades *from* releases earlier than UEC 3.2.0.0 (such as 3.1.0.x or 3.1.1.x) *to* releases 3.2.0.0 and later. Following the upgrade, UEC databases reside in the location specified by the user's currently configured `working_directory` location. This location defaults to "**\Program Files\Universal\UECtlr\bin**".

If the current UEC install was not an upgrade, it may be necessary to pass the path to the `uec_evm.db` file as a command line argument to the script. You can provide an absolute path or a path relative to the `uecdbrec.bat` script's location.

The user ID with which the recovery script runs requires appropriate permissions to the database directory and to the database file. Write access is required to the directory and read and write access is required to the database file.

The `uecdbrec.bat` batch file accepts an optional argument—the database file name to recover. If no database file name is specified, the `uec_evm.db` database is recovered. The batch file ends with exit code 0 if successful and a non-zero exit code if it failed.

---

## 14.6 Listing Infitran Database Records Examples

This section contains examples demonstrating the listing of Infitran database records via the Universal Spool List (USLIST) utility.

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

### Windows and UNIX

---

[List Universal Broker Database](#)

[List Universal Event Monitor Spool Database Records](#)

[List Broker Detail for a Component](#)

[List Standard Out for a Component](#)

## 14.6.1 List Universal Broker Database

---

[Figure 14.1](#) and [Figure 14.2](#), below, illustrate how to execute Universal Spool List (USLIST) with all defaults. (No options are required to issue USLIST.)

### Windows

```
cd c:\program files\universal\uspool\bin
uslist
```

Figure 14.1 Universal Spool List for Windows - List Universal Broker Database

### UNIX

```
cd /opt/universal/bin
uslist
```

Figure 14.2 Universal Spool List for UNIX - List Universal Broker Database

Since no USLIST options are supplied, this example, by default, lists the contents of the Universal Broker Component database (UBROKER). A summary of all records is produced; no detail component records are written.

The broker database must be located in the default directory.

### Components

#### Universal Spool List



## 14.6.2 List Universal Event Monitor Spool Database Records

Figure 14.3 and Figure 14.4, below, illustrate how to list the spool database records for a specific component (in this case, Universal Event Monitor).

### Windows

```
cd c:\program files\universal\uspool\bin
uslist -list uems
```

Figure 14.3 Universal Spool List for Windows - List Universal Event Monitor Spool Database Records

### UNIX

```
cd /opt/universal/bin
uslist -list uems
```

Figure 14.4 Universal Spool List for UNIX - List Universal Event Monitor Spool Database Records

These examples list the contents of the Universal Event Monitor spool database. A summary of all records is written.

### Command Line Options

The command line options used in this example are:

Option	Description
-list	Type of database from which to select record to write.
-ucmdspooldir	Directory location in which the Universal Command Server Component database ( <b>scomponent.db</b> ) is located.

### Components

#### Universal Spool List

## 14.6.3 List Broker Detail for a Component

Figure 14.5 and Figure 14.6, below, illustrate how to list the Broker detail for a specific component ID.

### Windows

```
cd c:\program files\universal\uspool\bin
uslist -component 123456789
```

Figure 14.5 Universal Spool List for Windows - List Broker Detail for a Component

### UNIX

```
cd /opt/universal/bin
uslist -component 123456789
```

Figure 14.6 Universal Spool List for UNIX - List Broker Detail for a Component

Since the **-list** option is not supplied, these examples, by default, list the contents of the Universal Broker Component database (UBROKER).

Because a component ID is specified, this will cause detail broker records to be written for component ID **123456789**.

### Command Line Options

The command line option used in this example is:

Option	Description
<b>-component</b>	Component identifier for which records will be selected to write.

### Components

#### Universal Spool List

## 14.6.4 List Standard Out for a Component

Figure 14.7 and Figure 14.8, below, illustrate how to list the standard output spool file for a specific component ID.

### Windows

```
cd c:\program files\universal\uspool\bin
uslist -list STDOUT -component 123456789
```

Figure 14.7 Universal Spool List for Windows - List Standard Out for a Component

### UNIX

```
cd /opt/universal/bin
uslist -list STDOUT -component 123456789
```

Figure 14.8 Universal Spool List for UNIX - List Standard Out for a Component

The standard output spool file is written for component **123456789**.

### Command Line Options

The command line options used in this example are:

Option	Description
-list	Type of database from which to select records to write.
-component	Component identifier for which records will be selected to write.

### Components

#### Universal Spool List

---

## 14.7 Removing Infitran Database Records

This section contains examples demonstrating the removal of Infitran database records via the Universal Spool Remove utility.

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

### Windows and UNIX

---

[Remove Component Records](#)

[Remove Component Records: Change Broker Database Directory](#)

## 14.7.1 Remove Component Records

[Figure 14.9](#) and [Figure 14.10](#), below, illustrate how to execute Universal Spool Remove (USLRM) with defaults.

### Windows

```
cd c:\program files\universal\uspool\bin
uslrm -component 123456789
```

Figure 14.9 Universal Spool Remove for Windows - Remove Component Records

### UNIX

```
cd /opt/universal/bin
uslrm -component 123456789
```

Figure 14.10 Universal Spool Remove for UNIX - Remove Component Records

The only required option is **-component** (the component ID; you can execute Universal Spool List (USLIST) utility to find specific component IDs).

All Stonebranch Solutions database records will be removed for component **123456789**.

### Command Line Options

The command line options used in this example are:

Command Options	Description
-component	Component identifier for which records will be removed.

### Components

#### Universal Spool Remove

## 14.7.2 Remove Component Records: Change Broker Database Directory

Figure 14.11 and Figure 14.12, below, illustrate how to execute Universal Spool Remove (USLRM) and specify a database directory other than the default.

### Windows

```
cd c:\program files\universal\uspool\bin
uslrm -component 123456789 -brokerspooldir "c:\program files\universal\spool2"
```

Figure 14.11 Universal Spool Remove for Windows - Remove Component Records

### UNIX

```
cd /opt/universal/bin
uslrm -component 123456789 -brokerspooldir "c:\program files\universal\spool2"
```

Figure 14.12 Universal Spool Remove for UNIX - Remove Component Records

All Stonebranch Solutions database records will be removed.

The **-brokerspooldir** option specifies the directory location in which the Universal Broker Component database is located. If the directory has spaces, it must be enclosed within double ( " ) quotation marks.

### Command Line Options

The command line options used in this example are:

Command Options	Description
<b>-component</b>	Component identifier for which records will be removed.
<b>-brokerspooldir</b>	Directory location in which the Universal Broker Component database ( <b>bcomponent.db</b> ) is located

### Components

#### Universal Spool Remove

# Fault Tolerance Implementation

---

## 15.1 Overview

For Infitran, fault tolerance is the capability of its Stonebranch Solutions components to recover or restart from an array of error conditions that occur in any large IT organization.

Errors occur as a result of human, software, or hardware conditions. The more resilient a product is to errors, the greater value it offers.

Currently, network fault tolerance is implemented in one Infitran component:

- Universal Data Mover

## 15.2 Network Fault Tolerance

UDM uses the TCP/IP protocol for communications over a data network. The TCP/IP protocol is a mature, robust protocol capable of re-sending packets and rerouting packets when network errors occur. However, data networks do have problems significant enough to prevent the TCP/IP protocol from recovering. As a result, the TCP/IP protocol terminates the connection between the application programs. Like any application using TCP/IP, UDM is subject to these network errors. Should they occur, a product can no longer communicate and must shutdown or restart. These types of errors normally show themselves as premature closes, connection resets, time-outs, or broken pipe errors.

UDM provides the ability to circumvent these types of errors with its Network Fault Tolerant protocol. By using the network fault tolerant protocol, UDM traps the connection termination caused by the network error and it reestablishes the network connections. Once connections are reestablished, processing automatically resumes from the location of the last successful message exchange. No program restarts are required and no data are lost.

The network fault tolerant protocol acknowledges and checkpoints successfully received and sent messages, respectively. The network fault tolerant protocol does reduce data throughput. Consequentially, the use of network fault tolerance should be carefully weighed in terms of increased execution time versus the probability of network errors and cost of such errors. For example, it may be easier to restart a program than to incur increased execution time.

When a network connection terminates, the UDM Manager will enter a network reconnect phase. In the reconnect phase, the Manager attempts to connect to the UDM Server and reestablish its network connections. The condition that caused the network error may persist for only seconds or days. The Manager will attempt Server reconnection for a limited amount of time (configured with the [RECONNECT\\_RETRY\\_COUNT](#) and [RECONNECT\\_RETRY\\_INTERVAL](#) options). These two options determine, respectively, how many reconnect attempts are made and how often they are made. After all attempts have failed, the manager ends with an error.

When a network connection terminates, the Server enters a disconnected state and waits for the Manager to reconnect. The user process continues running; however, if the user process attempts any I/O on the standard files, it will block. The Server waits for the Manager to reconnect for a period of time defined by the Manager's [RECONNECT\\_RETRY\\_COUNT](#) and [RECONNECT\\_RETRY\\_INTERVAL](#). Once that time has expired, the Server terminates the user process and exits.

UDM can request the use of the network fault tolerant protocol. If the Server does not support the protocol or is not configured to accept the protocol, the Manager continues without using the protocol.

The [NETWORK\\_FAULT\\_TOLERANT](#) option is used to request the protocol.



---

## 15.2.1 Open Retry

---

Open Retry is a type of fault tolerance used at the session-establishment level.

(Network fault tolerance is used from the time that a session has been fully established until the session has terminated.)

Open Retry is used during the establishment phase of a session. UDM tries to establish a session when the `open` command is issued. If the `OPEN_RETRY` option value is **yes**, and UDM fails to establish the session due to a network error, timeout, or the inability to start a transfer server, it will retry the `open` command based on the settings of the `OPEN_RETRY_COUNT` and `OPEN_RETRY_INTERVAL` options.

---

## 15.2.2 Component Management

---

In order to fully understand Universal Data Mover fault tolerant features, some understanding of how the Universal Broker manages components is necessary.

Universal Broker manages component startup, execution, and termination. The broker and its components have the ability to communicate service requests and status information between each other.

The Broker maintains a database of components that are active or have completed and waiting for restart or reconnection. The component information maintained by the broker determines the current state of the component. This state information is required by the broker to determine if a restart or reconnect request from a manager is acceptable or not. The broker's component information can be viewed with the Universal Query program.

One piece of component information maintained by the broker is the component's communication state. The communication state primarily determines what state the Universal Data Mover Server is in regarding its network connection with a manager and the completion of the user process and its associated spooled data.

The communication state values are described in [Table 15.1](#), below.

- **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 executing with either the network fault tolerant protocol, is restartable, or both.  Note: The server cannot tell if the manager is still executing or not since it cannot communicate with it.
ESTABLISHED	NO	NO	Server and manager are connected and processing normally. This state 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.

Table 15.1 Component Communication States

# Network Data Transmission

## 16.1 Overview

Distributed systems, such as Universal Command, communicate over data networks. All Stonebranch components communicate using the TCP/IP protocol. The UDP protocol is not used for any product data communication over a network.

Stonebranch Solutions can utilize either of two network protocols:

1. [Secure Socket Layer Protocol](#)

Secure Socket Layer version 3 (**SSLv3**) provides the highest level of security available. SSL is a widely used and accepted network protocol for distributed software applications that are required to address all aspects of secure data transfer on private and public networks.

All Stonebranch Solutions components (version 3.x and later) use **SSLv3**.

2. [Stonebranch Solutions Protocol](#)

Stonebranch Solutions version 2 (**UNVv2**) legacy protocol is provided for backward compatibility with Stonebranch Solutions versions earlier than 3.x.

To ensure backward compatibility, this protocol is still supported by version 3.x components.

The following sections discuss each of the protocols.

In addition to the network protocol used to transmit data, Stonebranch Solutions application protocol is discussed as well.

---

## 16.1.1 Secure Socket Layer Protocol

---

Stonebranch Solutions implement the SSL protocol using the OpenSSL library or the IBM z/OS System SSL library, available on the z/OS operating system. The most recent SSL standard is version 3. A subsequent version was produced, changing the name to Transport Layer Security version 1 (TLSv1). TLSv1 is the actual protocol used by Stonebranch Solutions. TLSv1 is more commonly referred to simply as SSL and the term SSL is used throughout the rest of this documentation to mean TLSv1, unless otherwise noted.

The SSL protocol addresses the major challenges of communicating securely over a potentially insecure data network. The following sections discuss the issue of data privacy and integrity, and peer authentication.

### *Data Privacy and Integrity*

People with sufficient technical knowledge and access to network resources can watch or capture data transmitting across the network. What they do with the data is up to them.

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

As data is transmitted over the network, it passes through media and hardware of unknown quality that may erroneously change bits of data without warning. Additionally, although data may be encrypted, there is nothing stopping a malicious person from changing the data while it is transmitted over the network. The changed data may or may not be detected by the recipient depending on what changed and how it is processed. It may be accepted as valid data, but the information it represents is now erroneous.

Data integrity must be protected from errors in transmission and malicious users. Data integrity checks insure that what was sent is exactly what is received by the recipient. Without integrity checks, there is no guarantee.

Encryption algorithms 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 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 MAC's 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 MAC's and can be used synonymously in most contexts.

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

Stonebranch Solutions 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.

[Table 16.1](#), below, identifies the supported SSL cipher suites.

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

Table 16.1 Supported SSL cipher suites

Stonebranch Solutions support 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 [Stonebranch Solutions Protocol \(UNVv2\)](#).

### Selecting an SSL Cipher Suite

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

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

## 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. Section [7.7 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>

---

## 16.1.2 Stonebranch Solutions Protocol

---

The Stonebranch Solutions protocol (**UNVv2**) is a proprietary protocol that securely and efficiently transports data across data networks. **UNVv2** is used in Stonebranch Solutions 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 MAC's 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.

## 16.1.3 Stonebranch Solutions Application Protocol

---

Stonebranch Solutions components use an application-layer protocol to exchange data messages. The protocol has the following characteristics:

- [Low-Overhead](#)
- [Secure](#)
- [Extensible](#)
- [Configurable Options](#)

The following sections refer to two categories of data transmitted by Stonebranch Solutions:

- Control data (or messages) consists of messages generated by Stonebranch Solutions components in order to communicate with each other. The user of the product has no access to the control data itself.
- 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:

- **ZLIB** method offers the highest compression ratios with highest CPU utilization.
- **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

The protocol is secure. All control data exchanged between Stonebranch Solutions components are encrypted with a unique session key and contain a MAC. The encryption prevents anyone from analyzing the message data and attempting to circumvent product and customer policies. Each session uses a different encryption key to prevent "play back" types of network attacks, where messages captured from a previous session are replayed in a new session. This applies to both network protocols: SSL and **UNVv2**.

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.

The data encryption options affect the application data being sent over the network. Special fields, such as passwords, are always encrypted. The encryption option cannot be turned off for such data.



## Extensible

The message protocol used between the Stonebranch Solutions components is extensible. New message fields can be added with each new release without creating product component incompatibilities. This permits different component versions to communicate with each other with no problems. This is a very important feature for distributed systems, since it is near impossible to upgrade hundreds of servers simultaneously.

New encryption and compression algorithms can be added in future releases without losing backward compatibility with older releases. After a network connection is made, connection options are negotiated between the two Stonebranch Solutions 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.

---

## 16.1.4 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 of the Stonebranch Solutions components:

### CODE\_PAGE

The CODE\_PAGE option specifies the code page translation table used to translate network data from and to the local code page for the system on which the program is executing.

A codepage table is text file that contain a two-column table. The table maps local single byte character codes to two-byte UNICODE character codes.

Code pages are located in the product National Language Support (NLS) directory or library. New code pages may be created and added to the NLS directory or library. The CODE\_PAGE option value is simply the name of the code page file without any file name extension if present.

### CTL\_SSL\_CIPHER\_LIST

The CTL\_SSL\_CIPHER\_LIST option specifies one or more SSL cipher suites that are acceptable to use for network communications on the control session, which is used for component internal communication.

The SSL protocol uses cipher suites to specify the combination of encryption and message digest algorithms used for a session. An ordered list of acceptable cipher suites can be specified in a most to least order of preference.

An example cipher suite list is RC4-MD5,RC4-SHA,AES128-SHA. The RC4-MD5 cipher suite is the most preferred and AES128-SHA is the least preferred.

When a manager and server first connect, they perform an SSL handshake. The handshake negotiates the cipher suite used for the session. The manager and server each have a cipher suite list and the first one in common is used for the session.

Why is a list of cipher suites helpful? A distributed software solution may cross many organizational and application boundaries each with their 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 may 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.

## DATA\_AUTHENTICATION

The DATA\_AUTHENTICATION option specifies whether or not the network data is authenticated. Data authentication verifies that the data did not change from the point it was sent to the point it was received.

Data authentication also is referred to as a data integrity in this document.

Data authentication occurs for each message sent over the network. If a message fails authentication, the network session is terminated and both programs end with an error.

The DATA\_AUTHENTICATION option is applicable to the UNVv2 protocol only. SSL always performs authentication.

## DATA\_COMPRESSION

The DATA\_COMPRESS 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:

- ZLIB method provides the highest compression ratio with the highest use of CPU
- HASP method provides the lowest compress ratio with the lowest use of CPU.

Whether or not compression is used and which compression method is used depends on several items:

- Network bandwidth. If network bandwidth is small, compression may be worth the cost in CPU.
- CPU resources. If CPU is limited, the CPU cost may not be worth the reduced bandwidth usage.
- Data compression ratio. If the data does not compress well, it is probably not worth CPU cost. If the data ratio is high, the CPU cost may be worth it.

## DATA\_ENCRYPTION

The DATA\_ENCRYPTION option specifies whether or not network data is encrypted.

Encryption translates data into a format that prevents the original data from being determined. Decryption translates encrypted data back into its original form.

The type of encryption performed depends on the network protocol being used, SSL or UNVv2.

Data encryption does increase CPU usage. Whether or not encryption is used depends on the sensitivity of the data and the security of the two host systems and the data network between the hosts.

## DATA\_SSL\_CIPHER\_LIST

The `DATA_SSL_CIPHER_LIST` option specifies one or more SSL cipher suites that are acceptable to use for network communications on the data session, which is used for standard I/O file transmission.

(See [CTL\\_SSL\\_CIPHER\\_LIST](#) in this section.)

## DEFAULT\_CIPHER

The `DEFAULT_CIPHER` option specifies the SSL cipher suite to use (since SSL protocol requires a cipher suite) if the [DATA\\_ENCRYPTION](#) option is set to `NO`. The default `DEFAULT_CIPHER` is `NULL-MD5` (no encryption, MD5 message digest).

All SSL cipher suites have a message digest for good reasons. The message digest ensures that the data sent are the data received. Without a message digest, it is possible for bits of the data packet to get changed without being noticed.

## 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  $2 \times \text{NETWORK\_DELAY}$  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  $\text{KEEPALIVE\_INTERVAL} + 2 \times \text{NETWORK\_DELAY}$ ).

## NETWORK\_DELAY

The `NETWORK_DELAY` option provides the ability to fine tune Stonebranch Solutions 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, Stonebranch Solutions 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. Stonebranch Solutions 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 Stonebranch Solutions components to build direct translation tables. Direct translation is the fastest translation method when a significant amount (greater than 10K) of text data is transmitted. The code page exchange increases the amount of data sent over the network as part of the network connection negotiation. UCS translation does not require the exchange of code pages. For transactions that have little text data transmission, this is the fastest.

Binary data is transmitted without any data translation.

---

# z/OS Cancel Command Support

---

## 17.1 Overview

Infitran network fault tolerance provides users with the ability to execute jobs that will continue to run when the network is down (see Chapter [15 Fault Tolerance Implementation](#)).

However, there are scenarios in which the user may want to cancel an executing job that supports network fault tolerance and have both the manager and server processes terminate immediately. Because of fault tolerance, when the manager is terminated, the server side would begin a connection reestablishment protocol and continue to execute. This would allow the started user job to continue running.

In particular, z/OS supports a CANCEL command that will terminate a job executing on the z/OS operating system. 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.

---

## 17.1.1 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.

---

## 17.1.2 Security Token

---

For security purposes, Stonebranch Solutions 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.

# Glossary

---

This glossary defines terms used within the Infitran business solution:

## Agent

A single Infitran (or Indesca) installation comprised of one Universal Broker and one or more Stonebranch Solutions components, such as Universal Data Mover.)

## API

API (Application Programming Interface) is a set of functions, procedures, methods, classes, or protocols that an operating system, library, or service provides to support requests made by computer programs.

## Asynchronous Communication

Transmission of data or sending of messages without the need to wait for a reply from the destination before continuing with the next operation.

## CA

CA (**C**ertification **A**uthority) is a trusted third-party organization that issues digital certificates used to create digital signatures and public-private key pairs, guaranteeing that the individual granted the unique certificate is, in fact, who he or she claims to be.

## Channel

Medium used to convey information from sender to receiver.

## Communications Protocol

Set of standard rules for data representation, signaling, authentication, and error detection required to send information over a communications channel.



### Connector

Component used to allow one system or application to communicate with another system or application. A connector can be embedded within an application or operate as a stand-alone component.

### Container

Application environment that provides a runtime environment that offers services such as security, authentication, transaction management, and deployment to an application developer, thus enabling a faster implementation and rollout.

### EAI tools

EAI (**E**nterprise **A**pplication **I**ntegration) tools are used for the unrestricted sharing of data and business processes throughout the networked applications or data sources in an organization.

### GLBA

GLBA (**G**ramm-**L**each-**B**iley **A**ct) is a law enabling the consolidation of commercial banks, investment banks, securities firms and insurance companies.

### HIPAA

HIPPA (**H**ealth **I**nsurance **P**ortability and **A**ccountability **A**ct) is a law that serves to protect health insurance coverage for workers and their families when they change or lose their jobs.

### HTTP

HTTP (**H**yper**T**ext **T**ransfer **P**rotocol), a synchronous request / reply protocol, is the underlying protocol used by the World Wide Web to define how messages are formatted and transmitted, and what actions web servers and browsers should take in response to various commands.

### Internet Application

A web application (webapp) that is accessed via a web browser over the Internet.

### Internet Workload

Internet workload is any application, service, or function that operates in an Internet environment, such as web applications or container applications, and supports an Internet-based communication protocol such as HTTP or SOAP.

## JMS

**JMS (Java Message Service)** is an API that provides a standard way for Java programs to access and interact with an enterprise asynchronous messaging system. JMS uses both point-to-point (queue-based) and publish / subscribe (topic-based) messaging patterns.

## JMS Connector

Component that allows the sending and receiving of JMS messages between applications.

## Light-Weight Container Architecture (LWCA)

This architecture, combined with the Federated architecture of the current Stonebranch Solutions line, provide your enterprise with a loosely coupled, scalable, and secure solution to your enterprise workload management tasks.

## Listen MEP

**Listen MEP (Message Exchange Pattern)** refers to a component that listens for a message from an application or service.

## Managed File Transfer

Software solutions that facilitate the secure transfer of data from one computer to another through a network, such as the Internet, while offering a higher level of security and control than FTP.

## Managers

Infitran component that provides client services initiating requests on behalf of the user (for example, a Universal Command manager batch job requesting the execution of a command on a remote server).

## Message

Abstract format, or container, for sending data between applications or services. No implementation is implied.

## Message-Based Application

A message-based application accesses a target application by sending a message to a queue that is controlled by the target application. This queue must be known and accessible to the application sending the message.

## Message-Based Workload

Any application, service, or function that supports a message-based communication protocol such as JMS or MQ.

### Message Exchange Pattern

A Message Exchange Pattern (MEP) describes the pattern of messages required by a communications protocol in order to establish or use a communication channel.

### Message-based application

Application accessed via a web browser over the Internet or Intranet.

### MQ Connector

MQ connector supports workload execution via the MQ messaging protocol using synchronous and asynchronous communication.

### PKI

PKI (**P**ublic **K**ey **I**nfrastructure) a system of digital certificates, **C**As, and other registration authorities that verify and authenticate the validity of each party involved in an Internet transaction.

### proxy certificates

Proxy certificate is a certificate that is derived from, and signed by, a normal X.509 Public Key End Entity Certificate or by another proxy certificate for the purpose of providing restricted proxying and delegation within a **PKI**-based authentication system.

### Publish MEP

Publish Message Exchange Pattern, or MEP, represents an asynchronous outbound workload execution event that sends a message from an application or service to a target destination. This means that you can request execution of a workload using the JMS protocol to a target JMS provider.

### Remote Procedure Call

Remote Procedure Call (RPC) is the most common messaging pattern in SOAP. In RPC, one network node (the client) sends a request message to another node (the server). The server immediately sends a response message to the client. This type of transaction also is known as "request / reply."

### Request / Reply MEP

Request / Reply MEP represents an outbound request to a target workload followed by an inbound reply from a target workload. This is a synchronous operation, as the calling party waits, or blocks, for the reply to come back before releasing its resources and moving on to the next task.

## SAP

SAP ("**S**ystem **A**nalysis and **P**rogramming **D**evelopment") is a corporation providing enterprise software applications and support to businesses. SAP ERP is its enterprise resource planning software for managing information and among all company functions.

## Servers

Infitran component initiated either by a client or the Universal Broker. All servers are started by the Universal Broker. A manager can request that the Broker initiates a server on its behalf, and the manager and server then work together to perform a service, or a server can be started automatically by the Broker when the Broker starts and stopped when the Broker stops.

## SOA

SOA (**S**ervice-**O**riented **A**rchitecture) provides methods for systems development and integration where systems group functionality around business processes and package these as interoperable services.

SOA also describes IT infrastructure, which allows different applications to exchange data with one another as they participate in business processes.

## SOAP

SOAP (**S**imple **O**bject **A**ccess **P**rotocol) is a lightweight XML-based messaging protocol used to encode the information in web service request and response messages before sending them over a network. SOAP messages can be transported using a variety of Internet protocols.

SOAP is used predominantly to provide an interface to web service-based workload or legacy workload with a web service interface.

## SOAP Connector

Component that allows the sending and receiving of SOAP messages.

## SOX

SOX (**S**arbanes-**O**xley Act) is a law enacted to ensure accurate financial reporting by public companies.

## SSL encryption

SSL encryption uses SSL (**S**ecure **S**ockets **L**ayer) protocol to encrypt private documents for transmission via the Internet. SSL uses two keys to encrypt data - a public key known to everyone and a private key known only to the recipient of the message.

## STDIN

**STDIN (standard in), STDOUT (standard out), and STDERR (standard error) are the standard data streams between a computer program and its environment.**

## X.509 certificates

**Digital certificate issued by a [CA](#) that is defined according to the X.509 standard for defining digital certificates.**

## Web Services Description Language (WSDL)

**WSDL is an XML-based language that provides a model for describing Web services. WSDL is often used in combination with SOAP and XML Schema to provide web services over the Internet.**

## WebSphere XD (Extended Deployment) Environment

**WebSphere is designed to set up, operate, and integrate e-business applications across multiple computing platforms using Java-based Web technologies.**

## Workload

**Jobs, processes, applications, and services that require execution, usually in a scheduler or automation-based environment.**

## XD Connector

**XD Connector supports workload execution within the WebSphere Extended Deployment environment using synchronous communication via the SOAP protocol.**

# Customer Support

---

Stonebranch, Inc. provides customer support, via telephone and e-mail, for all Infitran components.

## E-MAIL

---

### All Locations

[support@stonebranch.com](mailto:support@stonebranch.com)

Customer support contact via e-mail also can be made via the Stonebranch website:

[www.stonebranch.com](http://www.stonebranch.com)

## TELEPHONE

---

Customer support via telephone is available 24 hours per day, 7 days per week.

### North America

**(+1) 678 366-7887, extension 6**

**(+1) 877 366-7887, extension 6 [toll-free]**

### Europe

**+49 (0) 700 5566 7887**

---

# Index

---

## A

alerts *315*  
types *316*

## C

CA certificate *222*  
certificate  
CA (Certificate Authority) *222*  
creating *223*  
configuration  
remote *234*  
configuration files  
reading *250*  
Universal Broker *252*  
console application *274*  
creating  
CA certificate *222*  
certificate *223*

## E

error messages  
translating into return codes *305*  
event log (Windows)  
writing records *323*  
executing  
Universal Broker for Windows *274*

## F

files  
input to Universal Message Translator *305*

## I

IBM i  
enabling Universal Broker *278*  
ending Universal Broker *278*  
removing \*ALLOBJ authority *169*  
starting Universal Broker *278*

## M

managed mode *235*

## Q

querying Universal Broker *317*

## R

reading  
configuration files *250*  
remote configuration *234*  
managed mode *235*  
unmanaged mode *234*

## S

start-up modes *237*

## T

translating error messages into return codes  
*305*

## U

Universal Broker  
start-up *237*

---

---

## Universal Message Translator

input files 305

## UNIX

configuration refresh 259

console application 276

daemon process 275

databases 339

unmanaged mode 234

## W

### Windows

databases 340

environment 274

Universal Access Control List 167

writing event log records 323

Windows service 274

writing event log records (Windows) 323

## X

X.509 certificates 197

## Z

### z/OS

databases 339

starting Universal Broker 273

stopping Universal Broker 273







**950 North Point Parkway, Suite 200  
Alpharetta, Georgia 30005  
U.S.A.**

