



Stonebranch Solutions

Version 4.2.0

Indesca
User Guide
indesca-user-4201

Indesca

User Guide

Stonebranch Solutions 4.2.0

Document Name	Indesca 4.2.0 User Guide				
Document ID	indesca-user-4201				
Components	z/OS	UNIX	Windows	IBM i	HP NonStop
Universal Command	✓	✓	✓	✓	✓
Universal Command Agent or SOA		✓	✓		
Universal Connector	✓	✓			
Universal Enterprise Controller	✓		✓		
Universal Enterprise Controller Client Applications			✓		
Universal Event Monitor	✓	✓	✓		
Universal Event Monitor for SOA		✓	✓		
Universal Certificate	✓	✓	✓		
Universal Control	✓	✓	✓	✓	✓
Universal Copy		✓	✓	✓	✓
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.[®] 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 Indesca 4.2.0 User Guide
(indesca-user-4201)
October 29, 2010

- Created links from Stonebranch Solutions components to their corresponding reference guides in Section [1.4 Indesca Components](#).
- Replaced z/OS examples with Windows and UNIX examples in Chapter [14 Databases](#).

Changes for Indesca 4.2.0 User Guide
(indesca-user-4200)
August 6, 2010

- This is the first version of the Indesca 4.2.0 User Guide. Information on Indesca 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	25
List of Tables	33
Preface	35
Document Structure	35
Cross-Reference Links	35
Conventions	36
Document Information	38
1 Indesca Overview	40
1.1 What is Indesca?	40
1.2 What Can Indesca Do for Me?	42
1.3 Indesca Features	43
1.4 Indesca Components	45
Universal Command	45
Universal Command Agent for SOA	45
Universal Event Monitor	45
Universal Event Monitor for SOA	46
Universal Enterprise Controller	46
Universal Event Subsystem	46

Universal Enterprise Controller Client Applications	47
I-Activity Monitor	47
I-Management Console	47
I-Administrator	47
Universal Connector	47
Universal Broker	48
Stonebranch Solutions Utilities	48
Universal Certificate	48
Universal Control	48
Universal Copy	48
Universal Database Dump	48
Universal Database Load	48
Universal Display Log File	49
Universal Encrypt	49
Universal Event Log Dump	49
Universal Message Translator	49
Universal Install Merge	49
Universal Query	49
Universal Return Code	49
Universal Spool List	49
Universal Spool Remove	50
Universal Submit Job	50
Universal Write-to-Operator	50
2 Remote Execution	51
2.1 Overview	51
2.2 Execution Primer	53
2.2.1 Executing Universal Command Manager on z/OS	54
2.2.2 Executing Universal Command Manager on Windows	55
2.2.3 Executing Universal Command Manager on UNIX	56
2.2.4 Executing Universal Command Manager on IBM i	57
2.2.5 Executing Universal Command Manager on HP NonStop	58
2.3 Remote Execution Examples	59
z/OS	59
Windows	60
UNIX	60
IBM i	60
HP NonStop	61
2.3.1 Back up UNIX Directory to z/OS Dataset	62
2.3.2 Restore UNIX Directory Backup from z/OS Dataset to UNIX Directory ...	64

2.3.3	Directory Listing for UNIX Server from z/OS	66
2.3.4	Directory Listing for Windows Server from z/OS	67
2.3.5	Provide Network Status of Remote UNIX from z/OS	68
2.3.6	Use UNIX tee Command to Store stdout to Local Server and z/OS	69
2.3.7	Use an Encrypted Command File for User ID and Password	70
2.3.8	Override Standard I/O File ddnames	72
2.3.9	Override Standard Files with Procedure Symbolic Parameters	73
2.3.10	Specifying UCMD Options with the EXEC PARM	74
2.3.11	Execute an Existing Windows .bat file from z/OS	75
2.3.12	Using Manager Fault Tolerance from z/OS	76
2.3.13	Restarting a Manager Fault Tolerant UCMD Manager on z/OS	78
2.3.14	Automatically Create a Unique Command ID Using CA-Driver Variables	80
2.3.15	Automatically Create a Unique Command ID for your Indesca Process Using Zeke Variables	83
2.3.16	Automatically Create a Unique Command ID for your Indesca Process Using OPC Variables	84
2.3.17	Universal Submit Job from z/OS to IBM i using the Remote Reply Facility	85
2.3.18	Back up UNIX Directory to Windows	87
2.3.19	Restore UNIX Directory Backup from Windows to UNIX	88
2.3.20	Provide Network Status of Remote UNIX from Windows	90
2.3.21	Redirect Standard Out and Standard Error to Windows	91
2.3.22	Spawn Background Process with nohup: UNIX	92
2.3.23	Redirect Standard Input from Initiating System to Windows	93
2.3.24	Redirect Standard Input from /dev/null to Windows	94
2.3.25	Authenticate from Windows Using Encrypted Parameters	95
2.3.26	Universal Submit Job from Windows to IBM i	96
2.3.27	Executing Universal Return Code within a Script via Universal Command Manager for z/OS	97
2.3.28	Executing Universal Return Code and Universal Message Translator within a Script via Universal Command Manager for z/OS	98
2.3.29	Provide Network Status of Remote Windows from UNIX	99
2.3.30	Redirect Standard Out and Standard Error to UNIX	100
2.3.31	Redirect Standard Input from Initiating System to UNIX	101
2.3.32	Redirect Standard Input from /dev/null to UNIX	102
2.3.33	Universal Submit Job from UNIX to IBM i	103
2.3.34	Provide Network Status of Remote Windows from IBM i	104
2.3.35	Execute Script to Provide Network Status of Remote Windows from IBM i	105
2.3.36	Display Library with Manager Fault Tolerance Active Using USBMJOB	106

2.3.37	Universal Submit Job from z/OS to IBM i	108
2.3.38	Provide Network Status of Remote Windows from HP NonStop	110
2.3.39	Execute Script to Provide Network Status of Remote Windows from HP NonStop	111
3	Remote Execution for SAP Systems	112
3.1	Overview	112
3.1.1	Work Requests	113
3.2	Mass Activities Support in Universal Connector	114
3.2.1	Initiating Mass Activities	115
3.2.2	Monitoring Mass Activities	116
3.2.3	Working with Parameter Records	117
3.3	Mass Activities Support Examples	118
	z/OS	118
3.3.1	Universal Connector Mass Activity Example 1	119
3.4	Batch Input Monitoring in Universal Connector	124
3.4.1	Batch Input Monitoring Process	124
3.4.2	Batch Input Monitoring Requirements	125
	SAP System	125
	SAP Batch Input Sessions	125
	Universal Connector	125
3.4.3	Batch Input Monitoring Parameters	126
3.5	Batch Input Monitoring Examples	127
	z/OS	127
3.5.1	Batch Input Processing Example	128
3.6	Remote Execution for SAP Systems Examples	131
	z/OS and UNIX	131
	z/OS	131
	UNIX	132
3.6.1	Define Job, Run Job, Get Output, and Purge Job	133
3.6.2	Submitting a Job to an SAP System Using a Pre-existing SAP Job as a Template – z/OS	134
3.6.3	Submitting a Job to an SAP System Using a Universal Connector Job Definition File – z/OS	136
3.6.4	Running a Job on an SAP System Using a Pre-existing SAP Job – z/OS	138
3.6.5	Running a Job on an SAP System Using a Universal Connector Job Definition File – z/OS	140
3.6.6	Running an SAP Job on a Specific SAP Server – z/OS	142
3.6.7	Variant Substitution – z/OS	144

3.6.8	Creating a USAP Variant Definition Using the Universal Connector GENERATE VARDEF Command – z/OS	148
3.6.9	Creating a Universal Connector Job Definition Using the GENERATE JOBDEF Command – z/OS	150
3.6.10	Submitting a Job to an SAP System Using a Pre-existing SAP Job as a Template – UNIX	152
3.6.11	Submitting a Job to an SAP System Using a Universal Connector Job Definition File – UNIX	153
3.6.12	Running a Job on an SAP System Using a Pre-existing SAP Job – UNIX	154
3.6.13	Running a Job on an SAP System Using a Universal Connector Job Definition File – UNIX	156
3.6.14	Running an SAP Job on a Specific SAP Server – UNIX	158
3.6.15	Variant Substitution – UNIX	160
3.6.16	Creating a USAP Variant Definition Using the USAP GENERATE VARDEF Command – UNIX	163
3.6.17	Creating a USAP Job Definition Using the USAP GENERATE JOBDEF Command – UNIX	164
4	Web Services Execution	165
4.1	Overview	165
4.2	Outbound Implementation	166
4.3	Inbound Implementation	166
4.4	Web Services - Outbound Examples	167
	Windows and UNIX	167
	UNIX	167
4.4.1	Basic Structure of Using Indesca to Publish to a SOA Workload	168
4.4.2	Message Payload for SOAP	171
	SOAP Response	171
4.4.3	Logging Configuration	173
	uac_log4jConfiguration.xml Example	174
	uai_log4jConfiguration.xml Example	175
4.4.4	Outbound SOAP Implementation	176
4.5	Web Services - Inbound Examples	179
	Windows and UNIX	179
4.5.1	Inbound JMS Examples	180
	ActiveMQ Topic	181
	Websphere Queue	182
	MQ Series Queue:	183
	Triggering an Event	184
4.5.2	Inbound SOAP Implementation	185

5	Event Monitoring and File Triggering	190
5.1	Overview	190
5.2	Universal Event Monitor	191
5.2.1	Storing Event Definitions and Event Handlers	193
5.2.2	Monitoring a Single Event	195
5.2.3	Monitoring Multiple Events	197
5.3	UEMLoad	199
5.3.1	Controlling Database Access	200
	Access via UEMLoad Utility	200
	Universal Access Control List	201
5.4	Event Monitoring and File Triggering Examples	202
	Universal Event Monitoring Examples	202
	z/OS	202
	Windows	202
	UNIX	203
5.4.1	Starting an Event-Driven Server	204
5.4.2	Refreshing an Event-Driven UEM Server	205
5.4.3	Using a Stored Event Handler Record in z/OS	206
5.4.4	Handling an Event With a Script in z/OS	207
5.4.5	Handling an Expired Event in z/OS	209
5.4.6	Continuation Character - in z/OS Handler Script	210
5.4.7	Continuation Character + in z/OS Handler Script	211
5.4.8	Continuation Characters - and + in z/OS Handler Script	212
5.4.9	Using a Stored Event Handler Record in Windows	213
5.4.10	Executing a Script for a Triggered Event Occurrence in Windows ...	214
5.4.11	Handling an Expired Event in Windows	216
5.4.12	Adding a Single Event Record for Windows	217
5.4.13	Adding a Single Event Handler Record for Windows	218
5.4.14	Listing All Event Definitions for Windows	219
5.4.15	Exporting the Event Definition and Event Handler Databases for Windows	220
5.4.16	List a Single Event Handler Record for Windows	221
5.4.17	Listing Multiple Event Definitions and Event Handlers Using Wildcards for Windows	222
5.4.18	Add Record(s) Using a Definition File for Windows	223
5.4.19	Add Record(s) Remotely, Using a Definition File Redirected from STDIN for Windows	224
5.4.20	Add Record(s) Remotely, Using a Definition File Redirected from STDIN (for z/OS) for Windows	225
5.4.21	Definition File Format for Windows	226

5.4.22	Using a Stored Event Handler Record in UNIX	228
5.4.23	Executing a Script for a Triggered Event Occurrence in UNIX	229
5.4.24	Handling an Expired Event in UNIX	231
5.4.25	Adding a Single Event Record for UNIX	232
5.4.26	Adding a Single Event Handler Record for UNIX	233
5.4.27	Listing All Event Definitions for UNIX	234
5.4.28	Exporting the Event Definition and Event Handler Databases for UNIX	235
5.4.29	List a Single Event Handler Record for UNIX	236
5.4.30	Listing Multiple Event Definitions and Event Handlers Using Wildcards for UNIX	237
5.4.31	Add Record(s) Using a Definition File for UNIX	238
5.4.32	Add Record(s) Remotely, Using a Definition File Redirected from STDIN for UNIX	239
5.4.33	Add Record(s) Remotely, Using a Definition File Redirected from STDIN (for z/OS) for UNIX	240
5.4.34	Definition File Format for UNIX	241
6	Security	243
6.1	Overview	243
6.2	Security of Indesca Components	244
6.2.1	Universal Broker	245
	File Permissions	245
	Configuration Files	246
	Universal Access Control List	246
	Universal Broker User Account	247
6.2.2	Universal Command Manager	249
	File Permissions	249
	RACF Protection	249
	Configuration Files	250
6.2.3	Universal Command Server	251
	File Permissions	251
	Configuration Files	251
	Universal Command Server User ID	252
	User Authentication	252
	Universal Command Server User Profile	253
	Universal Command Server User ID	253
6.2.4	Universal Event Monitor Manager	254
	File Permissions	254
	Data Privacy	254

RACF Protection	254
Configuration Files	255
6.2.5 Universal Event Monitor Server	256
Data Privacy	256
File Permissions	256
Configuration Files	256
User Authentication	257
6.2.6 Universal Control Manager	258
File Permissions	258
Configuration Files	258
RACF Protection	258
6.2.7 Universal Control Server	259
File Permissions	259
Configuration Files	259
Universal Control Server User ID	259
User Authentication	260
6.2.8 Universal Event Log Dump	261
Event Log Access	261
Universal Configuration Manager	261
6.2.9 Universal Spool List	261
6.2.10 Universal Spool Remove	261
6.3 Encryption	262
6.3.1 Encrypting Password Files	262
6.3.2 Transferring Encrypted Password Files between Servers	263
Security Considerations	263
6.4 Encryption Examples	264
z/OS	264
Windows	264
UNIX	264
IBM i	264
HP NonStop	264
6.4.1 Creating Encrypted Files for z/OS	265
6.4.2 Using Encrypted Password File on z/OS	266
6.4.3 Creating Encrypted Files for Windows	267
6.4.4 Using Encrypted Password File on Windows	268
6.4.5 Creating Encrypted Files for UNIX	269
6.4.6 Using Encrypted Password File on UNIX	270
6.4.7 Creating Encrypted Files for IBM i	271
6.4.8 Using Encrypted Password File on IBM i	272
6.4.9 Creating Encrypted Files for HP NonStop	273

6.5 Universal Access Control List	274
6.5.1 UACL Configuration	275
6.5.2 UACL Entries	276
Client Identification	276
Request Identification	280
Certificate-Based and Non Certificate-Based UACL Entries	281
6.6 Universal Access Control List Examples	282
z/OS	282
Windows	282
UNIX	282
IBM i	282
HP NonStop	283
6.6.1 Universal Broker for z/OS	284
6.6.2 Universal Command Server for z/OS	285
6.6.3 Universal Control Server for z/OS	286
6.6.4 Universal Broker for Windows	287
6.6.5 Universal Command Server for Windows	288
6.6.6 Universal Control Server for Windows	289
6.6.7 Universal Broker for UNIX	290
6.6.8 Universal Command Server for UNIX	291
6.6.9 Universal Control Server for UNIX	292
6.6.10 Universal Broker for IBM i	293
6.6.11 Universal Command Server for IBM i	294
6.6.12 Universal Control Server for IBM i	295
6.6.13 Universal Broker for HP NonStop	296
6.6.14 Universal Command Server for HP NonStop	297
6.6.15 Universal Control Server for HP NonStop	298
6.7 X.509 Certificates	299
6.7.1 Sample Certificate Directory	300
6.7.2 Sample X.509 Certificate	301
6.7.3 Certificate Fields	302
6.7.4 SSL Peer Authentication	303
Certificate Verification	303
Certificate Revocation	303
Certificate Identification	304
Certificate Support	304
6.8 Creating Certificates Examples	305
6.8.1 Creating a Certificate Authority Certificate	306
6.8.2 Creating a Certificate	307
6.9 Command References	308

6.9.1 Command Reference Syntax	308
Options Section	308
Command Section	309
Command Reference Example	310
6.9.2 Configuration	310
6.9.3 Command Reference Request	311
Examples	311
7 Copying Files to / from Remote Systems	312
7.1 Overview	312
z/OS	312
Windows	313
UNIX	313
IBM i	314
HP NonStop	314
7.1.1 Copy from Local z/OS to Remote Windows via Universal Copy	315
7.1.2 Copy from Remote Windows to Local z/OS via Universal Copy	316
7.1.3 Copy from Local z/OS to Remote UNIX via Universal Copy	317
7.1.4 Copy from Remote UNIX to Local z/OS via Universal Copy	318
7.1.5 Copy from Local z/OS to Remote IBM i via Universal Copy	319
7.1.6 Copy from Remote IBM i to Local z/OS via Universal Copy	320
7.1.7 Copy from Local z/OS to Remote HP NonStop via Universal Copy	321
7.1.8 Copy from Remote HP NonStop to Local z/OS via Universal Copy	322
7.1.9 Third-Party Copy via Local z/OS, from Windows to UNIX via Universal Copy	323
7.1.10 Third-Party Copy via Local z/OS, from UNIX to Windows via Universal Copy	325
7.1.11 Third-Party Copy via Local z/OS, from Windows to Windows via Universal Copy	327
7.1.12 Third-Party Copy via Local z/OS, from UNIX to UNIX via Universal Copy	330
7.1.13 Copy from Local z/OS to Remote System (in Binary) via Universal Copy	332
7.1.14 Copy from Remote System to Local z/OS (in Binary) via Universal Copy	333
7.1.15 Copy from Local z/OS to Remote z/OS (with Encryption, Compression, and Data Authentication) via Universal Copy	334
7.1.16 Copy from Remote z/OS to Local z/OS (with Encryption, Compression, and Data Authentication) via Universal Copy	336
7.1.17 Copy Local File from Local z/OS to Remote Windows (with Windows Date Variables) via Universal Copy	338

7.1.18	Copy Local File from Local z/OS to Remote UNIX (with UNIX Data Variables) via Universal Copy	340
7.1.19	Copy File from Remote UNIX to Local z/OS Using UNIX cat Command via Universal Command Manager for z/OS	342
7.1.20	Command Coded as a Script, Script Stored on z/OS via Universal Command Manager for z/OS	343
7.1.21	Copy from Remote UNIX to Local Windows via Universal Command Manager for Windows	344
7.1.22	Copy From Local Windows to Remote UNIX via Universal Command Manager for Windows	345
7.1.23	Copy a File from Remote UNIX to Local Windows Using the UNIX cat Command via Universal Command Manager for Windows	346
7.1.24	Copy from Remote Windows to Local UNIX via Universal Command Manager for UNIX	347
7.1.25	Copy from Local UNIX to Remote Windows via Universal Command Manager for UNIX	348
7.1.26	Copying a File from Remote Windows to Local UNIX via Universal Command Manager for UNIX	349
7.1.27	Copying a File from Local UNIX to Remote Windows via Universal Command Manager for UNIX	350
7.1.28	Copy from Remote Windows to Local IBM i via Universal Command Manager for IBM i	351
7.1.29	Copy from Local IBM i to Remote Windows via Universal Command Manager for IBM i	352
7.1.30	Copy a File from Remote Windows to Local IBM i via Universal Command Manager for IBM i	353
7.1.31	Copy a File from Local IBM i to Remote Windows via Universal Command Manager for IBM i	354
7.1.32	Copy a File from Remote IBM i to Local Windows via Universal Command Manager for IBM i	355
7.1.33	Copy a File from Local Windows to Remote IBM i	356
7.1.34	Copy File from Remote Windows to Local IBM i via Universal Command Manager	357
7.1.35	Copy File from Remote Windows to Local HP NonStop via Universal Copy	358
7.1.36	Copy Local File from Local HP NonStop to Remote Windows via Universal Copy	359
7.1.37	Copy File from Remote Windows to Local HP NonStop via UCMD Manager for HP NonStop	360
7.1.38	Copy File from Remote Windows to Local HP NonStop via UCMD Manager for HP NonStop	361
7.1.39	Copy a File from Local HP NonStop to Remote Windows via UCMD Manager for HP NonStop	362

7.1.40	Copy a File from Local HP NonStop to Remote Windows via UCMD Manager for HP NonStop	363
8	Configuration Management	364
8.1	Overview	364
8.2	Configuration Methods	365
	Universal Broker / Servers Configuration Method	365
8.2.1	Command Line	366
8.2.2	Command Line File	368
8.2.3	Environment Variables	369
8.2.4	Configuration File	371
	Configuration File Syntax	373
8.3	Remote Configuration	374
8.3.1	Unmanaged Mode	374
8.3.2	Managed Mode	375
	Selecting Managed Mode	375
8.3.3	Universal Broker Start-up	377
8.4	Universal Configuration Manager	378
8.4.1	Availability	378
8.4.2	Accessing the Universal Configuration Manager	380
8.4.3	Navigating through Universal Configuration Manager	382
8.4.4	Modifying / Entering Data	382
	Rules for Modifying / Entering Data	382
8.4.5	Saving Data	383
8.4.6	Accessing Help Information	383
8.4.7	Universal Command Installed Components	384
	Universal Command Manager	384
	Universal Command Server	385
8.4.8	Universal Command Agent for SOA Installed Components	386
	Universal Application Container Server	386
8.4.9	Universal Event Monitor Installed Components	387
	Universal Event Monitor Manager	387
	Universal Event Monitor Server	388
8.4.10	Universal Enterprise Controller Component	389
8.4.11	Universal Broker Installed Component	390
8.5	Configuration Refresh	391
8.5.1	Configuration Refresh Via Universal Control	392
	Configuration Refresh for Universal Event Monitor Server	392
8.5.2	Configuration Refresh Via Universal Configuration Manager	393

8.5.3	Universal Broker Configuration Options Refresh	393
8.6	Refreshing via Universal Control Examples	394
z/OS	394
Windows	394
UNIX	394
IBM i	394
HP NonStop	394
8.6.1	Refreshing Universal Broker from z/OS	395
8.6.2	Refreshing a Component from z/OS	397
8.6.3	Refreshing Universal Broker via Universal Control from Windows	398
8.6.4	Refreshing a Component via Universal Control from Windows	399
8.6.5	Refreshing Universal Broker via Universal Control from UNIX	400
8.6.6	Refreshing a Component via Universal Control from UNIX	401
8.6.7	Refreshing Universal Broker via Universal Control from IBM i	402
8.6.8	Refreshing a Component via Universal Control from IBM i	403
8.6.9	Refreshing Universal Broker via Universal Control from HP NonStop	404
8.6.10	Refreshing a Component via Universal Control from HP NonStop	405
8.7	Merging Configuration Options during an Upgrade Installation Examples	406
Windows and UNIX	406
Files Used in Examples	407
8.7.1	Merge Files Using Program Defaults	408
8.7.2	Merge Files Introducing New Options	409
8.7.3	Merge Files Using Installation-Dependent Values	410
9	Component Management	411
9.1	Overview	411
9.2	Component Definition	412
9.2.1	Universal Event Monitor Component Definition	412
9.3	Starting Components	413
Starting Manually	413
Start via Manager	413
Starting Automatically	413
Starting via Universal Control	413
9.4	Stopping Components	414
9.5	Starting / Stopping Universal Broker Examples	415
z/OS	415
Windows	415
UNIX	415
IBM i	415

HP NonStop	415
9.5.1 Starting / Stopping Universal Broker for z/OS	416
Start Universal Broker	416
Stop Universal Broker	416
9.5.2 Starting Universal Broker for Windows	417
Console Application	417
Windows Service	417
9.5.3 Starting Universal Broker for UNIX	418
Daemon	418
Console Application	419
9.5.4 Starting, Ending, and Working With Universal Broker for IBM i	420
Commands	421
9.5.5 Starting Universal Broker for HP Nonstop	422
Console Application	422
Daemon	423
9.6 Starting / Stopping Components via Universal Control Examples	424
z/OS	424
Windows	424
UNIX	424
IBM i	424
HP NonStop	425
9.6.1 Starting a z/OS Component via Universal Control	426
9.6.2 Stopping a z/OS Component via Universal Control	427
9.6.3 Starting a Windows Component via Universal Control	428
9.6.4 Stopping a Windows Component via Universal Control	429
9.6.5 Starting a UNIX Component via Universal Control	430
9.6.6 Stopping a UNIX Component via Universal Control	431
9.6.7 Starting an IBM i Component via Universal Control	432
9.6.8 Stopping an IBM i Component via Universal Control	433
9.6.9 Stopping an HP NonStop Component via Universal Control	434
9.7 Maintaining Universal Broker Definitions in the Universal Enterprise	
Controller Database	435
z/OS and Windows	435
z/OS	435
Windows	435
9.7.1 List All Defined Brokers	436
9.7.2 Export a Specific Defined Broker	436
9.7.3 Export Events	436
9.7.4 Retrieve Archived File and Export	437

9.7.5	Delete a Specific Defined Broker	437
9.7.6	Add Specific Defined Broker via deffile	438
9.7.7	Add Existing Brokers to a Broker Group	439
9.7.8	Delete Existing Brokers to a Broker Group	439
9.7.9	Export Events into ARC Format for z/OS	440
9.7.10	Retrieve Archived File and Export into XML for z/OS	440
9.7.11	Export Events into ARC Format for Windows	441
9.7.12	Retrieve Archive File and Export into CSV for Windows	441
10	Messaging and Auditing	442
10.1	Overview	442
10.2	Messaging	443
10.2.1	Message Types	443
10.2.2	Message ID	444
10.2.3	Message Levels	444
10.2.4	Message Destinations	445
z/OS	Message Destinations	445
UNIX	Message Destinations	445
Windows	Message Destinations	446
IBM i	Message Destinations	446
HP NonStop	Message Destinations	446
10.3	Auditing	447
10.4	Creating Write-to-Operator Messages Examples	448
z/OS	448
10.4.1	USS UWTO for z/OS Console	449
10.4.2	USS UWTO for z/OS Console and Wait for Reply	450
11	Message Translation	451
11.1	Overview	451
11.2	Usage	452
11.2.1	Translation Table	452
Translation Table Format	452
Translation Table Fields	453
11.2.2	Matching Algorithm	453
11.3	Message Translation Examples	454
All Operating Systems	454
z/OS	454
Windows	454
UNIX	454

IBM i	454
HP NonStop	455
11.3.1 Universal Message Translator (Part 1)	456
11.3.2 Universal Message Translator (Part 2)	457
11.3.3 Execute Universal Message Translator from z/OS	458
11.3.4 Execute Universal Message Translator from z/OS Manager (in a Script with Table Housed on Remote Server)	459
11.3.5 Execute Universal Message Translator from z/OS Manager (in a Script with Table Housed on z/OS)	461
11.3.6 Execute Universal Message Translator from Windows	463
11.3.7 Execute Universal Message Translator from UNIX	464
11.3.8 Execute Universal Message Translator from IBM i	465
11.3.9 Execute Universal Message Translator from HP NonStop	466
12 Monitoring and Alerting	467
12.1 Monitoring of All Agents	468
12.1.1 Monitored Information	468
12.1.2 Polling	468
12.1.3 Alerts	468
12.2 Querying for Job Status and Activity	469
12.3 Querying for Job Status and Activity Examples	470
All Operating Systems	470
z/OS	470
UNIX and Windows	470
IBM i	470
HP NonStop	470
12.3.1 Universal Query Output	471
12.3.2 Universal Query for z/OS	472
12.3.3 Universal Query for UNIX and Windows	473
12.3.4 Universal Query for IBM i	474
12.3.5 Universal Query for HP NonStop	475
13 Windows Event Log Dump	476
13.1 Overview	476
13.2 Windows Event Log Dump Examples	477
Windows	477
13.2.1 Execute Universal Event Log Dump from z/OS Manager	478
13.2.2 Execute Universal Event Log Dump from a Windows Server	480

14 Databases	481
14.1 Overview	481
14.2 Component Information Database	482
14.2.1 Universal Spool	483
14.3 Universal Event Monitor Databases	484
14.3.1 Event Definition Database	485
14.3.2 Event Handler Database	486
14.3.3 Event Spool Database	487
14.3.4 Controlling UEM Database Access	488
14.4 Universal Enterprise Controller Databases	489
14.4.1 Database Files	489
14.4.2 Database Management	489
Automated Database Cleanup	489
Memory Management	490
14.5 Database Backup and Recovery	491
14.5.1 Database Backups	491
14.5.2 General Database Recovery Procedures	492
14.5.3 Database Recovery for Universal Broker	493
z/OS	493
UNIX	493
Windows	494
IBM i	494
14.5.4 Database Recovery for Universal Enterprise Controller	495
z/OS	495
Windows	496
14.6 Listing Indesca Database Records Examples	497
Windows and UNIX	497
14.6.1 List Universal Broker Database	498
Windows	498
UNIX	498
14.6.2 List Universal Command Server Database Records	499
Windows	499
UNIX	499
14.6.3 List Broker Detail for a Component	500
Windows	500
UNIX	500
14.6.4 List Standard Out for a Component	501
Windows	501
UNIX	501

14.7 Removing Indesca Database Records	502
Windows and UNIX	502
14.7.1 Remove Component Records	503
Windows	503
UNIX	503
14.7.2 Remove Component Records: Change Broker Database Directory ...	504
Windows	504
UNIX	504
15 Fault Tolerance Implementation	505
15.1 Overview	505
15.2 Network Fault Tolerance in Universal Command	506
15.2.1 Network Fault Tolerant Protocol	506
15.2.2 Universal Command Manager	507
15.2.3 Universal Command Server	507
15.3 Manager Fault Tolerance in Universal Command	508
15.3.1 Functionality	508
Command Identifier	509
Standard I/O Files	509
Requesting Restart	510
15.3.2 Component Management	517
15.4 Network Fault Tolerance in Universal Connector	519
15.4.1 Points of Failure	519
15.4.2 Network Fault Tolerance Configuration Parameters	520
15.5 Client Fault Tolerance - Universal Connector	521
15.5.1 Overview	521
15.5.2 Modes	522
Pre-XBP 2.0 Client Fault Tolerance (CFT)	522
Secure Client Fault Tolerance (Secure CFT)	523
15.5.3 Parameters	524
Client Fault Tolerance Target Host	524
Client Fault Tolerance Command Prefix	524
Secure Client Fault Tolerance Option	524
Client Fault Tolerance ABAP Program	525
15.5.4 Command ID Job Step	526
Pre-XBP 2.0 CFT Mode	526
Secure CFT Mode	526
15.5.5 Command Identifier	526
15.5.6 Requesting Restart	527

Controlling Auto Restart	527
15.5.7 Sample Command Lines For Working With Client Fault Tolerance ...	528
Working With Job Definition Files	528
Working With Pre-defined SAP Jobs	530
15.6 Implementing Fault Tolerance Examples	532
Windows	532
15.6.1 Implementing Manager Fault Tolerance for Windows	533
16 Network Data Transmission	534
16.1 Overview	534
16.1.1 Secure Socket Layer Protocol	535
Data Privacy and Integrity	535
Peer Authentication	537
16.1.2 Stonebranch Solutions Protocol	538
Data Privacy and Integrity	538
16.1.3 Stonebranch Solutions Application Protocol	539
Low-Overhead	539
Secure	539
Extensible	540
16.1.4 Configurable Options	541
17 z/OS Cancel Command Support	545
17.1 Overview	545
17.2 z/OS CANCEL Command Support in Universal Command	546
17.2.1 Exit Codes	546
17.2.2 Security Token	547
17.3 z/OS CANCEL Command Support in Universal Connector	548
17.3.1 Exit Codes	548
A Glossary	549
B Customer Support	555
Index	556

List of Figures

1	Indesca Overview	40
	Figure 1.1 Indesca Operating Systems Interface	41
2	Remote Execution	51
	Figure 2.1 Remote Execution Components: Indesca Manager and Server	52
	Figure 2.2 Executing UCMD Manager on z/OS	54
	Figure 2.3 Executing UCMD Manager on Windows	55
	Figure 2.4 Executing UCMD Manager on UNIX	56
	Figure 2.5 Requirements for Executing UCMD Manager for IBM i	57
	Figure 2.6 Executing UCMD Manager on HP NonStop	58
	Figure 2.7 UCMD Manager for z/OS – Back up UNIX Directory to z/OS Dataset	62
	Figure 2.8 UCMD Manager for z/OS – z/OS to UNIX Backup Restore from z/OS Dataset	64
	Figure 2.9 UCMD Manager for z/OS – Directory Listing for UNIX Server	66
	Figure 2.10 UCMD Manager for z/OS – Directory Listing for Windows Server	67
	Figure 2.11 UCMD Manager for z/OS – Netstat Command for UNIX	68
	Figure 2.12 UCMD Manager for z/OS - Redirect Standard Out to File and UCMD Manager	69
	Figure 2.13 UCMD Manager for z/OS - Create an Encrypted Command File	70
	Figure 2.14 UCMD Manager for z/OS - Use an Encrypted Command File	70
	Figure 2.15 UCMD Manager for z/OS - Override Standard Files with ddname	72
	Figure 2.16 UCMD for z/OS - Override Standard Files with Procedure Symbolic Parameters	73
	Figure 2.17 UCMD Manager for z/OS - Specifying UCMD Options with the EXEC PARM	74
	Figure 2.18 UCMD Manager for z/OS - Execute a Windows .bat File	75
	Figure 2.19 UCMD Manager for z/OS - Using Manager Fault Tolerance from z/OS	76
	Figure 2.20 UCMD Manager for z/OS - Override Restart Parameter in Command Line	78
	Figure 2.21 UCMD Manager for z/OS - CA-Driver Procedure	80
	Figure 2.22 UCMD Manager for z/OS - Call PROC DRVRUCMD from within UCMD SYSIN DD statement	81
	Figure 2.23 UCMD Manager for z/OS - Override Variable Value for REMOTEJOBNAME in UCMDSTEP	82

Figure 2.24	UCMD Manager for z/OS - Unique Command ID with Zeke	83
Figure 2.25	UCMD Manager for z/OS - Unique Command ID with OPC	84
Figure 2.26	Universal Submit Job - z/OS to IBM i with WTOR Support	85
Figure 2.27	UCMD Manager for Windows - Directory Backup Script File Example	87
Figure 2.28	UCMD Manager for Windows - Directory Backup Example	87
Figure 2.29	UCMD Manager for Windows - Directory Restore Script Example	88
Figure 2.30	UCMD Manager for Windows - Directory Backup Example	88
Figure 2.31	UCMD Manager for Windows - System Status Script File Example	90
Figure 2.32	UCMD Manager for Windows - System Status Example	90
Figure 2.33	UCMD Manager for Windows - Redirect Standard Out and Standard Error	91
Figure 2.34	UCMD Manager for Windows - Spawn Background Process with nohup: UNIX	92
Figure 2.35	UCMD Manager for Windows - Redirect Standard Input from Initiating System	93
Figure 2.36	UCMD Manager for Windows - Redirect Standard Input from /dev/null	94
Figure 2.37	UCMD Manager for Windows - Authentication Parameters from Encrypted File	95
Figure 2.38	Universal Submit Job – Windows to IBM i	96
Figure 2.39	Universal Return Code - Universal Command Manager for z/OS Executing URC within a Script 97	
Figure 2.40	Universal Command Manager for z/OS Executing URC and UMET within a Script	98
Figure 2.41	UCMD Manager for UNIX - Network Status Script File Example	99
Figure 2.42	UCMD Manager for UNIX - System Status Example	99
Figure 2.43	UCMD Manager for UNIX - Redirect Standard Out and Standard Error	100
Figure 2.44	UCMD Manager for UNIX - Redirect Standard Input from Initiating System	101
Figure 2.45	UCMD Manager for UNIX - Redirect Standard Input from /dev/null	102
Figure 2.46	Universal Submit Job – UNIX to IBM i	103
Figure 2.47	UCMD Manager for IBM i - Network Status Script File	104
Figure 2.48	UCMD Manager for IBM i - Network Status Script File Command	104
Figure 2.49	UCMD Manager for IBM i - Command Coded as a Script	105
Figure 2.50	UCMD Manager for IBM i - Display Library	106
Figure 2.51	Universal Submit Job - z/OS to IBM i	108
Figure 2.52	UCMD Manager for HP NonStop - Network Status Script File	110
Figure 2.53	UCMD Manager for HP NonStop - System Status Example	110
Figure 2.54	UCMD Manager for HP NonStop - Command Coded as a Script	111
3	Remote Execution for SAP Systems	112
Figure 3.1	USPVRMS – JCL (1 of 2)	120
Figure 3.2	USPVRMS – JCL (2 of 2)	121
Figure 3.3	USPJRMS – JCL	122
Figure 3.4	USPPRC – JCL Procedure	123
Figure 3.5	Batch Input Processing (1 of 2)	128
Figure 3.6	Batch Input Processing (2 of 2)	129
Figure 3.7	Submitting a Job to an SAP System Using a Pre-existing SAP Job as a Template – z/OS	134
Figure 3.8	Submitting a Job to an SAP System Using a USAP Job Definition File – z/OS	136
Figure 3.9	Running a Job on an SAP System Using a Pre-existing SAP Job – z/OS	138

Figure 3.10	Running a Job on an SAP System Using a Universal Connector Job Definition File – z/OS	140
Figure 3.11	Running an SAP Job on a Specific SAP Server – z/OS	142
Figure 3.12	Variant Substitution – z/OS (1 of 3)	144
Figure 3.13	Variant Substitution – z/OS (2 of 3)	145
Figure 3.14	Variant Substitution – z/OS (3 of 3)	146
Figure 3.15	Creating a USAP Variant Definition Using the USAP GENERATE VARDEF Command – z/OS	148
Figure 3.16	Creating a Universal Connector Job Definition Using the GENERATE JOBDEF Command – z/OS	150
Figure 3.17	Submitting a Job to an SAP System Using a Pre-existing SAP Job as a Template – UNIX	152
Figure 3.18	Submitting a Job to an SAP System Using a Universal Connector Job Definition File – UNIX	153
Figure 3.19	Submitting a Job to an SAP System - Command to Submit Job – UNIX	153
Figure 3.20	Running a Job on an SAP System Using a Pre-existing SAP Job – UNIX	154
Figure 3.21	Running a Job on an SAP System - USAP Job Definition File	156
Figure 3.22	Running a Job on an SAP System - Command to Run Job	156
Figure 3.23	Running a Job on a Specific SAP Server - USAP Job Definition File	158
Figure 3.24	Running a Job on a Specific SAP Server - Command to Run Job	158
Figure 3.25	Variant Substitution for Variant SBT1- Variant Definition File	161
Figure 3.26	Variant Substitution for Variant SBT1 - Command Line	161
Figure 3.27	Variant Substitution for Variant SBT2- Variant Definition File	161
Figure 3.28	Variant Substitution for Variant SBT2 - Command Line	162
Figure 3.29	Command Line to Run a New Job using Variant Substitution	162
Figure 3.30	Creating a USAP Variant Definition Using the USAP GENERATE VARDEF Command	163
Figure 3.31	Creating a USAP Job Definition Using the USAP GENERATE JOBDEF Command	164
4	Web Services Execution	165
Figure 4.1	Basic Structure of Using Indesca to Publish to a SOA Workload	168
Figure 4.2	JMS ActiveMQ Workload	169
Figure 4.3	JMS Websphere Workload	169
Figure 4.4	XD Workload	170
Figure 4.5	MQ Series Workload	170
Figure 4.6	Message Payload - SOAP	171
Figure 4.7	Message Payload Response - SOAP	171
Figure 4.8	uac_log4jConfiguration.xml Example	174
Figure 4.9	uai_log4jConfiguration Example	175
Figure 4.10	Outbound SOAP Request - JCL	176
Figure 4.11	Outbound SOAP Request - SYSIN DD Contents	177
Figure 4.12	Outbound SOAP Request - SYSIN DD Contents	178
Figure 4.13	Outbound SOAP Request - SYSIN DD Contents	178
Figure 4.14	Inbound JMS - Constructing Connection to Target	180
Figure 4.15	Inbound JMS - Attachment to an Apache ActiveMQ Dynamic Topic	181
Figure 4.16	Inbound JMS - Attachment to an IBM Websphere Queue	182
Figure 4.17	Inbound JMS - Attachment to an IBM MQ Series Queue	183
Figure 4.18	Triggering an Event	184

Figure 4.19	Inbound SOAP Request UAC.xml	185
Figure 4.20	Inbound SOAP Request – Message Payload Written to process_%Seq%.xml File	186
Figure 4.21	Inbound SOAP Request – Universal Event Monitor Event Definition	186
Figure 4.22	Inbound SOAP Request – Universal Event Monitor Handler Definition	187
Figure 4.23	Outbound SOAP Request – abc.rexx	188
Figure 4.24	Outbound SOAP Request – Event and Handler to purge abc.log	189
5	Event Monitoring and File Triggering	190
Figure 5.1	High-Level Interaction of UEM Components	192
Figure 5.2	UEMLoad Utility Overview	194
Figure 5.3	UEM Manager Overview	196
Figure 5.4	UEM Server Overview	198
Figure 5.5	Starting a UEM Event-Driven Server	204
Figure 5.6	Refreshing a UEM Event-Driven Server	205
Figure 5.7	Using a Stored Event Record	206
Figure 5.8	Handling an Event with a Script	208
Figure 5.9	Handling an Expired Event	209
Figure 5.10	Continuation Character - in z/OS Handler Script	210
Figure 5.11	Continuation Character + in z/OS Handler Script	211
Figure 5.12	Continuation Characters - and + in z/OS Handler Script	212
Figure 5.13	Using a Stored Event Handler Record	213
Figure 5.14	Handling an Event with a Script	214
Figure 5.15	Contents of Sample Script File	214
Figure 5.16	Handling an Expired Event	216
Figure 5.17	Adding a single event definition record.	217
Figure 5.18	Adding a Single Event Handler Record	218
Figure 5.19	Listing all Event Definition Records	219
Figure 5.20	Exporting all Event and Handler Records	220
Figure 5.21	List a Single Event Handler Record	221
Figure 5.22	Sample List Output	221
Figure 5.23	Using Wildcards to List Records	222
Figure 5.24	Add Database Record(s) Using a Definition File	223
Figure 5.25	Redirect Definition File from stdin	224
Figure 5.26	Redirect Definition File from STDIN (for z/OS)	225
Figure 5.27	Definition File Sample - Windows	227
Figure 5.28	Using a Stored Event Handler Record	228
Figure 5.29	Handling an Event with a Script	229
Figure 5.30	Contents of Sample Script File	230
Figure 5.31	Handling an Expired Event	231
Figure 5.32	Adding a single event definition record.	232
Figure 5.33	Adding a single event handler record.	233
Figure 5.34	Listing all event definition records.	234
Figure 5.35	Exporting all Event and Handler Records	235
Figure 5.36	List a Single Event Handler Record	236
Figure 5.37	Sample List Output	236
Figure 5.38	Using Wildcards to List Records	237
Figure 5.39	Add Database Record(s) Using a Definition File	238
Figure 5.40	Redirect Definition File from stdin	239

Figure 5.41	Redirect Definition File from STDIN (for z/OS)	240
Figure 5.42	Definition File Sample - UNIX	242
6	Security	243
Figure 6.1	z/OS encrypted file example	266
Figure 6.2	Windows encrypted file example	268
Figure 6.3	UNIX encrypted file example	270
Figure 6.4	IBM i encrypted file example	272
Figure 6.5	Universal Configuration Manager - Universal Broker - Access ACL	287
Figure 6.6	Universal Configuration Manager - Universal Command Server - Access ACL	288
Figure 6.7	Universal Configuration Manager - Universal Control Server - Access ACL	289
Figure 6.8	X.500 Directory (sample)	300
Figure 6.9	X.509 Version 3 Certificate (sample)	301
Figure 6.10	Command Reference Syntax	310
7	Copying Files to / from Remote Systems	312
Figure 7.1	Copy from Local z/OS to Remote Windows via Universal Copy	315
Figure 7.2	Copy from Remote Windows to Local z/OS via Universal Copy	316
Figure 7.3	Copy from Local z/OS to Remote UNIX via Universal Copy	317
Figure 7.4	Copy from Remote UNIX to Local z/OS via Universal Copy	318
Figure 7.5	Copy from Local z/OS to Remote IBM i via Universal Copy	319
Figure 7.6	Copy from Remote IBM i to Local z/OS via Universal Copy	320
Figure 7.7	Universal Copy for z/OS - Copy from Local z/OS to Remote HP NonStop via Universal Copy	321
Figure 7.8	Copy from Remote HP NonStop to Local z/OS via Universal Copy	322
Figure 7.9	Third-Party Copy via Local z/OS, from Windows to UNIX via Universal Copy	323
Figure 7.10	Third-Party Copy via Local z/OS, from UNIX to Windows via Universal Copy	325
Figure 7.11	Third-Party Copy via Local z/OS, from Windows to Windows via Universal Copy	328
Figure 7.12	Third-Party Copy via Local z/OS, from UNIX to UNIX via Universal Copy	330
Figure 7.13	Copy from Local z/OS to Remote System (in Binary) via Universal Copy	332
Figure 7.14	Copy from Remote System to Local z/OS (in Binary) via Universal Copy	333
Figure 7.15	Copy from Local z/OS to Remote z/OS (with Encryption, Compression, and Data Authentication) via Universal Copy	334
Figure 7.16	Copy from Remote z/OS to Local z/OS (with Encryption, Compression, and Data Authentication) via Universal Copy	336
Figure 7.17	Copy from Local z/OS File to Remote Windows (with Windows Date Variables) via Universal Copy	338
Figure 7.18	Copy from Local z/OS File to Remote UNIX (with UNIX Date Variables) via Universal Copy	340
Figure 7.19	UNIX Copy to Local z/OS Dataset via Universal Command Manager for z/OS	342
Figure 7.20	Command Coded as a Script via Universal Command Manager for z/OS	343
Figure 7.21	Copy from Remote UNIX to Local Windows via Universal Command Manager for Windows	344
Figure 7.22	Copy from Local Windows to Remote UNIX via Universal Command Manager for Windows	345

Figure 7.23	Copy from Remote UNIX to Windows using UNIX cat Command via UCMD Manager for Windows	346
Figure 7.24	Copy from Remote Windows to Local UNIX via Universal Command Manager for UNIX	347
Figure 7.25	Copy from Local UNIX to Remote Windows via Universal Command Manager for UNIX	348
Figure 7.26	Copy from Remote Windows to UNIX via Universal Command Manager for UNIX	349
Figure 7.27	Copy from UNIX to Remote Windows via Universal Command Manager for UNIX	350
Figure 7.28	Copy from Remote Windows to Local IBM i via Universal Command Manager for IBM i	351
Figure 7.29	Copy from Local IBM i to Remote Windows via Universal Command Manager for IBM i	352
Figure 7.30	Copy from Remote Windows to IBM i via Universal Command Manager for IBM i	353
Figure 7.31	Copy from IBM i to Remote Windows via Universal Command Manager for IBM i	354
Figure 7.32	Copy from Remote IBM i to Windows via Universal Command Manager for IBM i	355
Figure 7.33	Copy from Windows to Remote IBM i Using Universal Copy on IBM i	356
Figure 7.34	Copy from Remote Windows to Local IBM i via Universal Command Manager for IBM i	357
Figure 7.35	Copy from Remote Windows to Local File via Universal Copy for HP NonStop	358
Figure 7.36	Copy Local File to Remote Windows via Universal Copy for HP NonStop	359
Figure 7.37	Copy from Remote Windows to HP NonStop via Universal Command Manager for HP NonStop	360
Figure 7.38	Copy Remote Windows File to Local HP NonStop via Universal Command Manager for HP NonStop	361
Figure 7.39	Copy from HP NonStop to Remote Windows via Universal Command Manager for HP NonStop	362
Figure 7.40	Copy Local File from to Remote Windows via Universal Command Manager for HP NonStop	363
8	Configuration Management	364
Figure 8.1	Remote Configuration - Unmanaged and Managed Modes of Operation	376
Figure 8.2	Universal Configuration Manager Error dialog – Windows Vista / Windows 7	378
Figure 8.3	Program Compatibility Assistant – Windows Vista / Windows 7	379
Figure 8.4	Universal Configuration Manager	381
Figure 8.5	Universal Configuration Manager - UCMD Manager	384
Figure 8.6	Universal Configuration Manager - UCMD Server	385
Figure 8.7	Universal Configuration Manager - UAC Server	386
Figure 8.8	Universal Configuration Manager - UEM Manager	387
Figure 8.9	Universal Configuration Manager - UEM Server	388
Figure 8.10	Universal Configuration Manager - Universal Enterprise Controller	389
Figure 8.11	Universal Configuration Manager - Universal Broker	390
Figure 8.12	Refreshing Universal Broker via Universal Control from z/OS	395

Figure 8.13	Refreshing Component via Universal Control from z/OS	397
Figure 8.14	Refreshing Universal Broker via Universal Control from Windows	398
Figure 8.15	Refreshing Component via Universal Control from Windows	399
Figure 8.16	Refreshing Universal Broker via Universal Control from UNIX	400
Figure 8.17	Refreshing Component via Universal Control from UNIX	401
Figure 8.18	Refreshing Universal Broker via Universal Control from IBM i	402
Figure 8.19	Refreshing Component via Universal Control from IBM i	403
Figure 8.20	Refreshing Universal Broker via Universal Control from HP NonStop	404
Figure 8.21	Refreshing Component via Universal Control from HHP NonStop	405
Figure 8.22	Merge infile.txt into outfile.txt using program defaults	408
Figure 8.23	Merge infile.txt into outfile.txt keeping new options	409
Figure 8.24	Merge infile.txt into outfile.txt using installation-dependent values	410
9	Component Management	411
Figure 9.1	Universal Broker for UNIX - Daemon Startup Script Syntax	418
Figure 9.2	Universal Broker for IBM i - Subsystem Start Command	421
Figure 9.3	Universal Broker for IBM i - Subsystem End Command	421
Figure 9.4	Universal Broker for IBM i - Subsystem Work With Command	421
Figure 9.5	Universal Broker for HP NonStop - Start Command	422
Figure 9.6	Universal Broker for HP NonStop - Daemon Startup Script Syntax	423
Figure 9.7	Universal Control for z/OS - Start Component Example	426
Figure 9.8	Universal Control for z/OS - Stop Example	427
Figure 9.9	Universal Control for Windows - Start Component Example	428
Figure 9.10	Universal Control for Windows - Stop Component Example	429
Figure 9.11	Start Component Example.	430
Figure 9.12	Universal Control Manager for UNIX - Stop Component Example 1	431
Figure 9.13	Start Component Example	432
Figure 9.14	Universal Control for IBM i - Stop Component Example	433
Figure 9.15	Universal Control Manager for HP NonStop - Stop Component Example 1	434
Figure 9.16	UECLoad - List All Defined Brokers	436
Figure 9.17	UECLoad - Export a Specific Defined Broker	436
Figure 9.18	UECLoad - Export Events	436
Figure 9.19	UECLoad - Retrieve Archived File and Export	437
Figure 9.20	UECLoad - Delete a Specific Defined Broker	437
Figure 9.21	UECLoad - Add Specific Defined Broker via a Definition File	438
Figure 9.22	UECLoad - Definition File used for Adding Specific Defined Broker	438
Figure 9.23	UECLoad - Add Existing Brokers to a Broker Group	439
Figure 9.24	UECLoad - Delete Existing Brokers to a Broker Group	439
Figure 9.25	UECLoad for z/OS - Export Events into ARC Format	440
Figure 9.26	UECLoad for z/OS- Retrieve Archived File and Export into XML	440
Figure 9.27	UECLoad for Windows - Export Events into ARC Format	441
Figure 9.28	UECLoad for Windows - Retrieve Archived File and Export into CSV	441
10	Messaging and Auditing	442
Figure 10.1	Universal WTO - Issue WTO to z/OS Console	449
Figure 10.2	Universal WTO - Issue WTOR to z/OS Console	450
11	Message Translation	451

Figure 11.1	Universal Message Translator - Example 1, Message File	456
Figure 11.2	Universal Message Translator - Example 1, Translation Table 1	456
Figure 11.3	Universal Message Translator - Example 1, Translation Table 2	456
Figure 11.4	Universal Message Translator - Example 2, Message File	457
Figure 11.5	Universal Message Translator - Example 2, Translation Table 1	457
Figure 11.6	Universal Message Translator - Execute from z/OS	458
Figure 11.7	Universal Message Translator - Execute from z/OS Manager (with Table on Remote Server)	459
Figure 11.8	Universal Message Translator - Execute from z/OS Manager (with Table on z/OS)	461
Figure 11.9	Universal Message Translator - Execute from Windows	463
Figure 11.10	Universal Message Translator - Execute from UNIX	464
Figure 11.11	Universal Message Translator - Execute from IBM i	465
Figure 11.12	Universal Message Translator - Execute from HP NonStop	466
12	Monitoring and Alerting	467
Figure 12.1	Universal Query Output	471
Figure 12.2	Universal Query for z/OS - Listing Active Components	472
Figure 12.3	Universal Query for UNIX and Windows - Listing Active Components	473
Figure 12.4	Universal Query for IBM i (specific port) - Listing Active Components	474
Figure 12.5	Universal Query for IBM i (default port) - Listing Active Components	474
Figure 12.6	Universal Query for HP NonStop - Listing Active Components	475
13	Windows Event Log Dump	476
Figure 13.1	Universal Event Log Dump - Execution from z/OS Manager	478
Figure 13.2	Universal Event Log Dump - Execution from Windows Server	480
14	Databases	481
Figure 14.1	Universal Spool List for Windows - List Universal Broker Database	498
Figure 14.2	Universal Spool List for UNIX - List Universal Broker Database	498
Figure 14.3	Universal Spool List for Windows - List Universal Server Database Records	499
Figure 14.4	Universal Spool List for UNIX - List Universal Server Database Records	499
Figure 14.5	Universal Spool List for Windows - List Broker Detail for a Component	500
Figure 14.6	Universal Spool List for UNIX - List Broker Detail for a Component	500
Figure 14.7	Universal Spool List for Windows - List Standard Out for a Component	501
Figure 14.8	Universal Spool List for UNIX - List Standard Out for a Component	501
Figure 14.9	Universal Spool Remove for Windows - Remove Component Records	503
Figure 14.10	Universal Spool Remove for UNIX - Remove Component Records	503
Figure 14.11	Universal Spool Remove for Windows - Remove Component Records	504
Figure 14.12	Universal Spool Remove for UNIX - Remove Component Records	504
15	Fault Tolerance Implementation	505
Figure 15.1	Case Example 1	511
Figure 15.2	Case Example 2	513
Figure 15.3	Case Example 3	515
Figure 15.4	UCMD Manager for Windows - Manager Fault Tolerance	533

List of Tables

Preface	35
Table P.1 Command Line Syntax	36
2 Remote Execution	51
Table 2.1 Remote Execution Primer Examples – Configuration Options	53
6 Security	243
Table 6.1 Certificate Map Matching Criteria	278
Table 6.2 Certificate Identifier Field	278
Table 6.3 Client IP Address - Matching Criteria	279
Table 6.4 Request Fields	281
Table 6.5 Certificate Fields	302
Table 6.6 Command Reference Options	309
8 Configuration Management	364
Table 8.1 UNIX Configuration File Directory Search	372
Table 8.2 Stonebranch Solutions Configuration File Sample (infile.txt)	407
Table 8.3 Stonebranch Solutions Configuration File Sample (outfile.txt)	407
Table 8.4 Contents of outfile.txt after default merge	408
Table 8.5 Contents of outfile.txt when keeping unmatched destination values	409
Table 8.6 Contents of outfile.txt when using installation-dependent values	410
9 Component Management	411
Table 9.1 Universal Broker - Command Line Arguments to Daemon Startup Script	418
Table 9.2 Universal Broker for HP NonStop - Command Line Arguments to Daemon Startup Script	423
11 Message Translation	451
Table 11.1 Universal Message Translator – Translation Table	453
Table 11.2 Universal Message Translator for IBM i - Return Codes	453

15	Fault Tolerance Implementation	505
	Table 15.1 Component Communication States	518
16	Network Data Transmission	534
	Table 16.1 Supported SSL cipher suites	536

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
bold monospace font	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.
BOLD UPPER CASE	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 Information

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

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

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 Indesca component(s) that form the solution illustrated by that example.

- [Indesca Overview](#) (Chapter 1)
Overview of the Indesca business solution.
- [Remote Execution](#) (Chapter 2)
Workload Execution user scenarios for the features presented in Chapter 2.
- [Configuration Management](#) (Chapter 3)
Description and examples of the Configuration feature.
- [Web Services Execution](#) (Chapter 4)
Description and examples of the Web Services Execution feature.
- [Event Monitoring and File Triggering](#) (Chapter 5)
Description and examples of the Event Monitoring and Triggers feature.
- [Security](#) (Chapter 6)
Description and examples of the Security feature.
- [Copying Files to / from Remote Systems](#) (Chapter 7)
Description and examples of the Copying Files to / from Remote Systems 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 Querying for Job Status and Activity 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 Indesca.
- [Customer Support](#) (Appendix B)
Customer support contact information for Indesca.

Indesca Overview

1.1 What is Indesca?

Indesca (**I**ndependent **S**cheduling **A**gent) is the Stonebranch Inc. business solution for job scheduling.

One of the most fundamental functions of an operating system is to provide a means for user interaction. User interfaces take many forms; the most common types are command line and graphical. The command line interface is provided by virtually all general-purpose operating systems today.

Indesca provides a command line interface that extends to any operating system that can be reached on the computer network. That is, the remote operating system's command line interface is extended to the local operating system's command line interface. The remote and local systems can be running two different operating systems. The Indesca interface is operating-system independent.

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

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

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

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

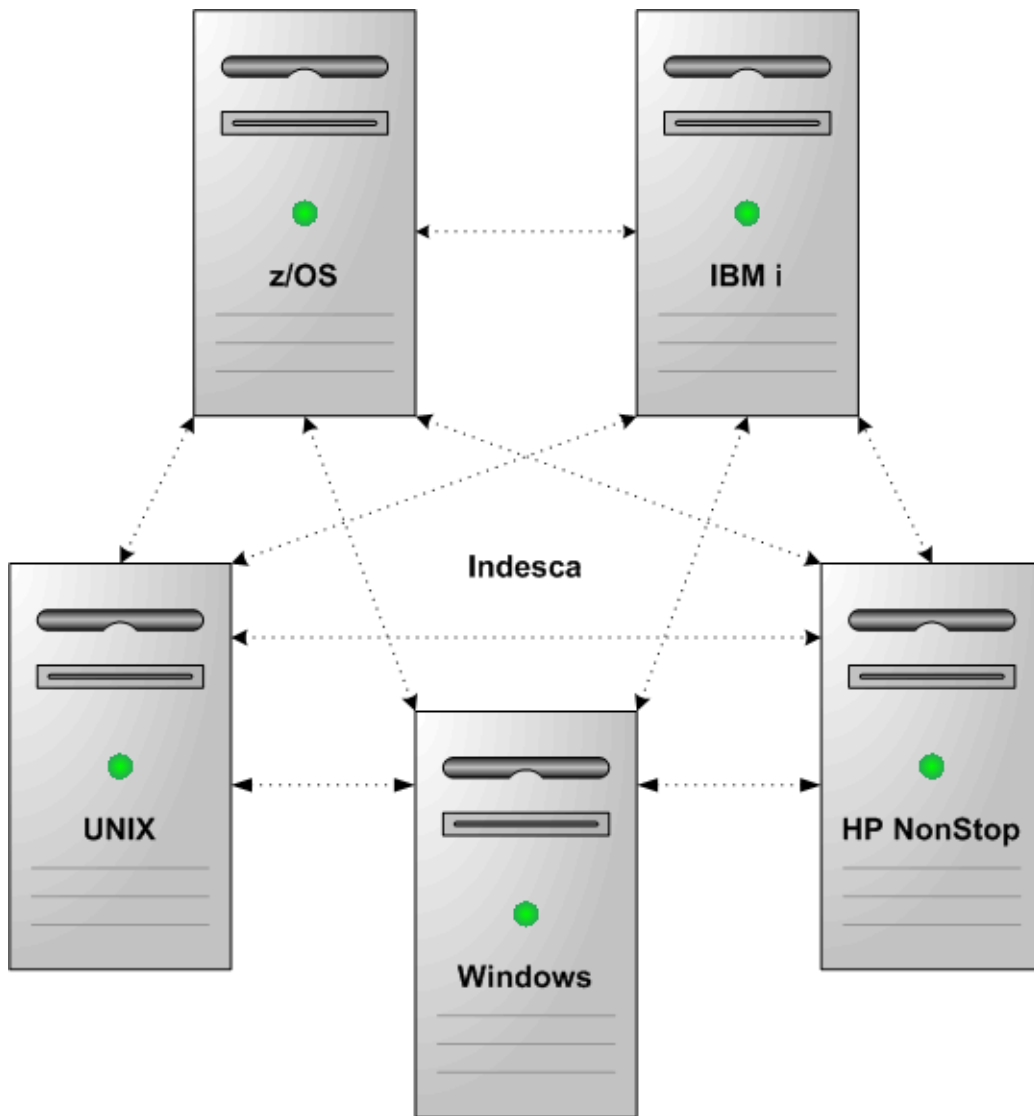


Figure 1.1 Indesca Operating Systems Interface

1.2 What Can Indesca Do for Me?

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

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

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

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

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

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

1.3 Indesca Features

The features that make Indesca 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 [Indesca Components](#) used in that implementation.

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

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

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

[Web Services Execution](#) enables Indesca to extend its remote execution functionality to Internet and message-based workload and create file-based events from inbound Internet and message-based application messages.

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

Indesca also provides a command line interface for [Copying Files to / from Remote Systems](#), whether from manager to server or server to manager.

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

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

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

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

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

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

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

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

1.4 Indesca Components

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

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

Universal Command

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

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

Universal Command Agent for SOA

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

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

Universal Event Monitor

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

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

Universal Event Monitor for SOA

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

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

Universal Enterprise Controller

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

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

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

Universal Event Subsystem

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

Universal Enterprise Controller Client Applications

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

I-Activity Monitor

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

I-Management Console

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

I-Administrator

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

Universal Connector

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

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

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

Universal Broker

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

Stonebranch Solutions Utilities

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

Universal Certificate

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

Universal Control

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

Universal Copy

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

Universal Database Dump

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

Universal Database Load

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

Universal Display Log File

Universal Display Log File 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 Indesca installation to query any broker in the Stonebranch infrastructure.

Universal Return Code

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

Universal Spool List

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

Universal Spool Remove

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

Universal Submit Job

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

Remote Execution

2.1 Overview

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

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

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

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

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

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

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

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

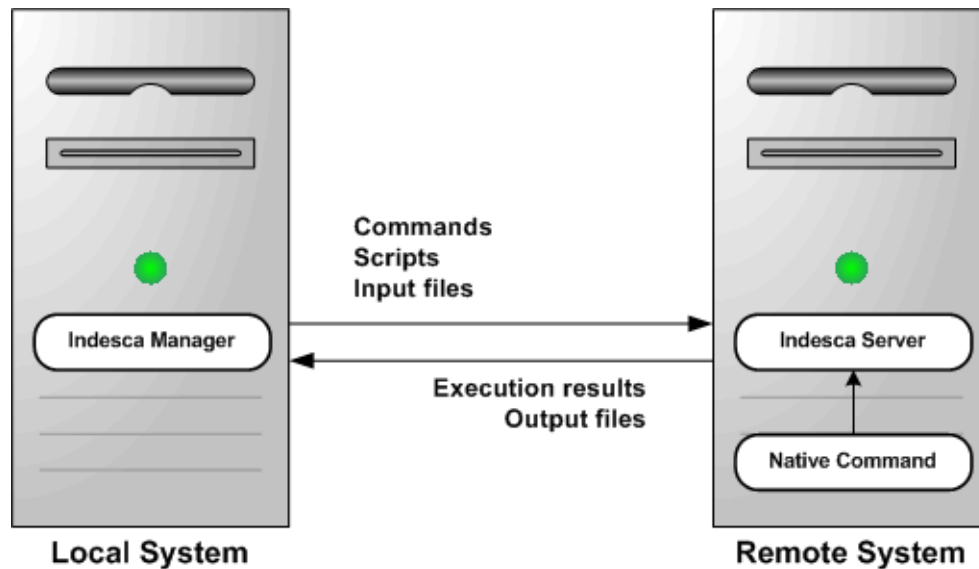


Figure 2.1 Remote Execution Components: Indesca Manager and Server

2.2 Execution Primer

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

Prior to reading this section, Section [2.1 Overview](#) should be read, as this section builds upon the material presented in the Overview. The primer discussions are from the perspective of the initiating system where the Universal Command (UCMD) Manager component is executed.

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

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

The following table describes each of the options used in the primer examples:

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

Table 2.1 Remote Execution Primer Examples – Configuration Options

2.2.1 Executing Universal Command Manager on z/OS

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

A UCMD Manager JCL procedure is provided with the Stonebranch Solutions installation to simplify JCL requirements. The JCL procedure name is **UCMDPRC**; it is located in the **SUNVSAMP** product library. See the [Universal Command Reference Guide](#) for more information on the UCMDPRC procedure.

[Figure 2.2](#), below, illustrates the JCL to execute UCMD Manager in a step. The input options are specified on the SYSIN ddname.

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

Figure 2.2 Executing UCMD Manager on z/OS

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

- Change the host name **da11as** specified with the **-host** option to the host name of the remote system on which to execute the command.
- Change the user ID **jo~~e~~** to a valid user ID on the remote system.
- Change the password value **abcde~~fg~~** to the password for the user ID.

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

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

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

Components

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

2.2.2 Executing Universal Command Manager on Windows

Universal Command Manager is run as a command on Windows.

Figure 2.3, below, illustrates the command and command line options to execute UCMD Manager.

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

Figure 2.3 Executing UCMD Manager on Windows

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

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

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

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

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

Components

Universal Command Manager for Windows

2.2.3 Executing Universal Command Manager on UNIX

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

Figure 2.4, below, illustrates the command and command line options to execute UCMD Manager.

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

Figure 2.4 Executing UCMD Manager on UNIX

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

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

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

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

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

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

Components

[Universal Command Manager for UNIX](#)

2.2.4 Executing Universal Command Manager on IBM i

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

Figure 2.5, below, illustrates the CL command and parameters to execute UCMD Manager.

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

Figure 2.5 Requirements for Executing UCMD Manager for IBM i

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

- Change the host name **da11as** specified with the HOST option to the host name of the remote system on which to execute the command.
- Change the user ID **joe** to a valid user ID on the remote system.
- Change the password value **abcdefg** to the password for the user ID.

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

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

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

Components

Universal Command Manager for IBM i

2.2.5 Executing Universal Command Manager on HP NonStop

Universal Command Manager is run as a TAACL command.

Figure 2.6, below, illustrates the command and command line options to execute UCMD Manager.

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

Figure 2.6 Executing UCMD Manager on HP NonStop

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

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

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

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

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

Components

Universal Command Manager for HP NonStop

2.3 Remote Execution Examples

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

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

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

z/OS

[Back up UNIX Directory to z/OS Dataset](#)

[Restore UNIX Directory Backup from z/OS Dataset to UNIX Directory](#)

[Directory Listing for UNIX Server from z/OS](#)

[Directory Listing for Windows Server from z/OS](#)

[Provide Network Status of Remote UNIX from z/OS](#)

[Use UNIX tee Command to Store stdout to Local Server and z/OS](#)

[Use an Encrypted Command File for User ID and Password](#)

[Override Standard I/O File ddnames](#)

[Override Standard Files with Procedure Symbolic Parameters](#)

[Specifying UCMD Options with the EXEC PARM](#)

[Execute an Existing Windows .bat file from z/OS](#)

[Using Manager Fault Tolerance from z/OS](#)

[Restarting a Manager Fault Tolerant UCMD Manager on z/OS](#)

[Automatically Create a Unique Command ID Using CA-Driver Variables](#)

[Automatically Create a Unique Command ID for your Indesca Process Using Zeke Variables](#)

[Automatically Create a Unique Command ID for your Indesca Process Using OPC Variables](#)

[Universal Submit Job from z/OS to IBM i using the Remote Reply Facility](#)

Windows

[Back up UNIX Directory to Windows](#)

[Restore UNIX Directory Backup from Windows to UNIX](#)

[Provide Network Status of Remote UNIX from Windows](#)

[Redirect Standard Out and Standard Error to Windows](#)

[Spawn Background Process with nohup: UNIX](#)

[Redirect Standard Input from Initiating System to Windows](#)

[Redirect Standard Input from /dev/null to Windows](#)

[Authenticate from Windows Using Encrypted Parameters](#)

[Universal Submit Job from Windows to IBM i](#)

[Executing Universal Return Code within a Script via Universal Command Manager for z/OS](#)

[Executing Universal Return Code and Universal Message Translator within a Script via Universal Command Manager for z/OS](#)

UNIX

[Provide Network Status of Remote Windows from UNIX](#)

[Redirect Standard Out and Standard Error to UNIX](#)

[Redirect Standard Input from Initiating System to UNIX](#)

[Redirect Standard Input from /dev/null to UNIX](#)

[Universal Submit Job from UNIX to IBM i](#)

IBM i

[Provide Network Status of Remote Windows from IBM i](#)

[Execute Script to Provide Network Status of Remote Windows from IBM i](#)

[Display Library with Manager Fault Tolerance Active Using USBMJOB](#)

[Universal Submit Job from z/OS to IBM i](#)

HP NonStop

[Provide Network Status of Remote Windows from HP NonStop](#)

[Execute Script to Provide Network Status of Remote Windows from HP NonStop](#)

2.3.1 Back up UNIX Directory to z/OS Dataset

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

The backup script is allocated to the ddname **MYSCRIPT**. The script writes the **tar .Z** file to its standard out which is transmitted by the UCMD Server to the UCMD Manager which writes it to the UNVOUT ddname. The data set **h1q.BKUP.TAR.Z** allocated to the UNVOUT ddname will contain the **tar .Z** backup file.

The data set **h1q.BKUP.TAR.Z** must be a variable record format data set. Any valid record length or valid block size will work.

```
//jobname JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
/*
//          JCLLIB ORDER=SBI.UNV.SUNVSAMP
/*
//STEP1    EXEC UCMDPRC,
//          STDOUT='DISP=SHR,DSN=h1q.BKUP.TAR.Z'
/*
//MYSCRIPT DD *
cd /export/home/username/fnd || exit 8
tar -cvzf - . || exit 8
//SYSIN    DD *
  -host     hostname
  -userid   username
  -pwd      password
  -script   MYSCRIPT
  -stdout   -mode binary
/*
```

Figure 2.7 UCMD Manager for z/OS – Back up UNIX Directory to z/OS Dataset

SYSIN Options

The SYSIN options used in this example are:

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

Components

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

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

This example demonstrates using UCMD Manager on z/OS to restore a directory on a UNIX system from a tar.Z backup maintained on the z/OS system. Refer to [Figure 2.7](#) to see how the backup data set was created.

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

```
//jobname JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*
// JCLLIB ORDER=SBI.UNV.SUNVSAMP
//*
//STEP1 EXEC UCMDPRC,
// STDIN='DISP=SHR,DSN=h7q.BKUP.TAR.Z'
//MYSCRIPT DD *
# Check if the directory exists. If it does not, create it.
  if test ! -d /export/home/username/fnd
    then mkdir /export/home/username/fnd || exit 8
  fi
  cd /export/home/username/fnd || exit 8
# Note: Not all tar commands recognize the 'B' argument. If you
# receive an error message indicating this from the remote
# UNIX system, remove the 'B' argument.
# The 'B' argument is used to force tar to read multiple
# times to fill the block.

tar -xzvBf - || exit 8

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

Figure 2.8 UCMD Manager for z/OS – z/OS to UNIX Backup Restore from z/OS Dataset

SYSIN Options

The SYSIN options used in this example are:

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

Components

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

2.3.3 Directory Listing for UNIX Server from z/OS

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

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

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

Figure 2.9 UCMD Manager for z/OS – Directory Listing for UNIX Server

SYSIN Options

The SYSIN options used in this example are:

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

Components

Universal Command Manager for z/OS

2.3.4 Directory Listing for Windows Server from z/OS

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

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

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

Figure 2.10 UCMD Manager for z/OS – Directory Listing for Windows Server

SYSIN Options

The **SYSIN** options used in this example are:

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

Components

Universal Command Manager for z/OS

2.3.5 Provide Network Status of Remote UNIX from z/OS

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

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

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

Figure 2.11 UCMD Manager for z/OS – Netstat Command for UNIX

SYSIN Options

The SYSIN options used in this example are:

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

Components

Universal Command Manager for z/OS

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

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

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

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

Figure 2.12 UCMD Manager for z/OS - Redirect Standard Out to File and UCMD Manager

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>-host</code>	Host name or IP address of the remote system on which to execute the script.
<code>-userid</code>	Remote user ID with which to execute the command.
<code>-pwd</code>	Password for the user ID.

Components

Universal Command Manager for z/OS

2.3.7 Use an Encrypted Command File for User ID and Password

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

Figure 2.13 demonstrates how to create a UENCRYPTED command file. The options to be encrypted are specified on the UNVIN ddname. In this example, the options to be encrypted are `-userid` and `-pwd`. The encrypted command file is written to ddname UNVOUT, which allocates PDS member **USR001**.

```
//jobname JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*
//STEP1 EXEC UENCRYPT
//STEPLIB DD DISP=SHR,DSN=SBI.UNV.SUNVLOAD
//UNVIN DD *
-userid username
-pwd password
//UNVOUT DD DISP=SHR,DSN=h7q.PROD.DATA(USR001)
//UNVNLS DD DISP=SHR,DSN=SBI.UNV.SUNVNLS
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
//SYSIN DD DUMMY
```

Figure 2.13 UCMD Manager for z/OS - Create an Encrypted Command File

Figure 2.14 demonstrates how to use an encrypted command file to execute a remote command using UCMD Manager. The example is executing a UNIX `ls` command on a remote system. The user ID and password to be used to execute the command is specified with the `-encryptedfile` option. The `-encryptedfile` option specifies a ddname from which to read the encrypted command file created in Figure 2.13.

```
//jobname JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*
// JCLLIB ORDER=SBI.UNV.SUNVSAMP
//*
//STEP1 EXEC UCMDPRC
//USER DD DISP=SHR,DSN=h7q.PROD.DATA(USR001)
//SYSIN DD *
-cmd 'ls -la'
-host hostname
-encryptedfile USER
/*
```

Figure 2.14 UCMD Manager for z/OS - Use an Encrypted Command File

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

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

SYSIN Options

The SYSIN options used in this example are:

Option	Description
<code>-cmd</code>	Command to be executed on the remote system.
<code>-host</code>	Host name or IP address of the remote system on which to execute the script.
<code>-encryptedfile</code>	ddname from which to read an encrypted command file created with the Stonebranch UENCRYPT utility.

Components

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

2.3.8 Override Standard I/O File ddnames

This example demonstrates how to override the standard output and error ddnames in the UCMD Procedure. The example overrides the UNVOUT ddname in the UCMDPRC procedure with a data set allocation and overrides the UNVERR ddname with a SYSOUT class of H.

```
//jobname JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*
// JCLLIB ORDER=SBI.UNV.SUNVSAMP
//*
//STEP1 EXEC UCMDPRC
//UNVOUT DD DISP=SHR,DSN=h7q.APP.LIST(OUTPUT)
//* UNIVERSAL COMMAND WILL CREATE THE MEMBER
//UNVERR DD SYSOUT=H
//SYSIN DD *
-host hostname
-userid username
-pwd password
-cmd command
/*
```

Figure 2.15 UCMD Manager for z/OS - Override Standard Files with ddname

SYSIN Options

The SYSIN options used in this example are:

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

Components

Universal Command Manager for z/OS

2.3.9 Override Standard Files with Procedure Symbolic Parameters

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

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

```
//jobname JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
/*
//          JCLLIB ORDER=SBI.UNV.SUNVSAMP
/*
//STEP1 EXEC UCMDPRC,
//  STDIN='DUMMY',
//  STDERR='SYSOUT=H',
//  STDOUT='DISP=SHR,DSN=h7q.DATA.LIST(OUT2)'
//SYSIN DD *
-host      hostname
-userid    username
-pwd       password
-cmd       command
/*
```

Figure 2.16 UCMD for z/OS - Override Standard Files with Procedure Symbolic Parameters

SYSIN Options

The SYSIN options used in this example are:

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

Components

Universal Command Manager for z/OS

2.3.10 Specifying UCMD Options with the EXEC PARM

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

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

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

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

Figure 2.17 UCMD Manager for z/OS - Specifying UCMD Options with the EXEC PARM

SYSIN Options

The SYSIN options used in this example are:

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

Components

Universal Command Manager for z/OS

2.3.11 Execute an Existing Windows .bat file from z/OS

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

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

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

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

Figure 2.18 UCMD Manager for z/OS - Execute a Windows .bat File

SYSIN Options

The SYSIN options used in this example are:

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

Components

Universal Command Manager for z/OS

2.3.12 Using Manager Fault Tolerance from z/OS

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

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

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

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

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

Figure 2.19 UCMD Manager for z/OS - Using Manager Fault Tolerance from z/OS

See [Chapter 15 Fault Tolerance Implementation](#) for complete details on the Manager Fault Tolerant feature.

SYSIN Options

The SYSIN options used in this example are:

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

Components

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

2.3.13 Restarting a Manager Fault Tolerant UCMD Manager on z/OS

In Section [2.3.12 Using Manager Fault Tolerance from z/OS](#), the example illustrated in [Figure 2.19](#) demonstrates how to remotely execute work using the Manager Fault Tolerant (MFT) feature.

This example ([Figure 2.20](#), below) demonstrates how to restart a UCMD Manager that prematurely ended when using MFT.

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

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

Continuing from [Figure 2.19](#), the example below illustrates a UCMD Manager restart for the work identified by command ID `example-2010-07-02`.

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

Figure 2.20 UCMD Manager for z/OS - Override Restart Parameter in Command Line

SYSIN Options

The SYSIN options used in this example are:

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

Components

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

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

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

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

- &C-L2JN is the CA-7 Job Name
- &C_L27# is the CA-7 Job Number
- &C_L2DOD is the Due Out Date for this Job execution.

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

Figure 2.21, below, illustrates an example of a CA-Driver Procedure using the above system variables to create a unique command ID.

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

Figure 2.21 UCMD Manager for z/OS - CA-Driver Procedure

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

(See the examples on the following pages.)

Example 1

Figure 2.22, below, illustrates how to call the PROC **DRVRUCMD** from within the UCMD **SYSIN DD** statement.

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

- **&C-L2JN** = PRD00001
- **&C_L27#** = 0030001
- **&C_L2DOD** 03265
- **&REMOTEJOBNAME** = UCMD (default value in Driver Procedure **DRVRUCMD**)

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

Expanded Results:

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

Figure 2.22 UCMD Manager for z/OS - Call PROC **DRVRUCMD** from within UCMD **SYSIN DD** statement

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

Components

Universal Command Manager for z/OS

Example 2

Figure 2.23, below, illustrates how to override the variable value for **REMOTEJOBNAME** in the **CALL** step.

```
//S1 EXEC UCMDPRC
//SYSIN DD *
//CALL EXEC PROC=DRVRUCMD,REMOTEJOBNAME=unixpayrolljob1
-cmd command -host hostname -userid username -pwd password
-managerft yes -restart auto
/*
Expanded Results:
//S1 EXEC UCMDPRC
//SYSIN DD *
-cmdid unixpayrolljob1PRD000010030001032625
-cmd command -host hostname -userid username -pwd password
-managerft yes -restart auto
/*
```

Figure 2.23 UCMD Manager for z/OS - Override Variable Value for REMOTEJOBNAME in UCMDSTEP

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

Components

Universal Command Manager for z/OS

2.3.15 Automatically Create a Unique Command ID for your Indesca Process Using Zeke Variables

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

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

```
//S1 EXEC UCMDPRC
//LOGONDD DD DISP=SHR,DSN=h1q.encrypted.file
//SCRIPTDD DD *
DIR
//SYSIN DD *
-cmddid 'jobname$SCHDATEunixpayrolljob1' <== Zeke variable
-script scriptdd -host dallas -encryptedfile logondd -managerft yes
-restart auto
/*
```

Expanded Results:

```
//S1 EXEC UCMDPRC
//LOGONDD DD DISP=SHR,DSN=h1q.encrypted.file
//SCRIPTDD DD *
dir
//SYSIN DD *
-cmddid 'JOBNAME03254unixpayrolljob1'
-script scriptdd -host dallas -encryptedfile logondd -managerft yes
-restart auto
/*
```

Figure 2.24 UCMD Manager for z/OS - Unique Command ID with Zeke

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

Components

Universal Command Manager for z/OS

2.3.16 Automatically Create a Unique Command ID for your Indesca Process Using OPC Variables

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

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

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

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

```
//S1 EXEC UCMDPRC
//LOGONDD DD DISP=SHR,DSN=h1q.encrypted.file
//SCRIPTDD DD *
DIR
//SYSIN DD *
-cmdid payrolljob&CYMD.&CHHMMSSX.            <== OPC variables
-script scriptdd -host dallas -encryptedfile logondd -managerft yes
-restart no
/*
```

Expanded Results:

```
//S1 EXEC UCMDPRC
//LOGONDD DD DISP=SHR,DSN=h1q.encrypted.file
//SCRIPTDD DD *
dir
//SYSIN DD *
-cmdid payrolljob2003061613315614
-script scriptdd -host dallas -encryptedfile logondd -managerft yes
-restart no
/*
```

Figure 2.25 UCMD Manager for z/OS - Unique Command ID with OPC

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

Components

Universal Command Manager for z/OS

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

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

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

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

```
//S1 EXEC UCMDPRC
//SCRIPT DD *
ADDLIBLE LIB(UNVPRD420)
UNVPRD420/USBMJOB CMD(dsp1ib ibmi-username) +
RMTRPY(*YES) +
RMTREFRESH(60) +
RMTMSGPRFX('TESTPRFX') +
RMTHOST(zos-hostname) +
MSGCMDPATH("/usr/local/universal/bin/uwto") +
RMTUSER(zos-username) + RMPWD(zos-password)
//SYSIN DD *
-script SCRIPT -host ibmi-hostname -userid ibmi-username -pwd ibmi-password
/*
```

Figure 2.26 Universal Submit Job - z/OS to IBM i with WTOR Support

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

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

The first line of the script will add the library **UNVPRD420** to the library concatenation. The second line will execute the command **dsp1ib ibmi-username** with the **USBMJOB** utility. All output created by the command will be spooled to stdout of the manager job.

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

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

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

SYSIN Options

The SYSIN options used in this example are:

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

USBMJOB Options

The USBMJOB options used in this example are:

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

Components

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

[Universal Submit Job](#)

2.3.18 Back up UNIX Directory to Windows

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

Figure 2.27, below, illustrates the script file.

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

Figure 2.27 UCMD Manager for Windows - Directory Backup Script File Example

Figure 2.28, below, illustrates the command to execute the script file.

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

Figure 2.28 UCMD Manager for Windows - Directory Backup Example

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

Command Line Options

The command line options used in this example are:

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

Components

Universal Command Manager for Windows

2.3.19 Restore UNIX Directory Backup from Windows to UNIX

This example restores the directory that was backed up in the previous example. The file containing the backup is on the local system.

Figure 2.29, below, illustrates the script to perform the restore.

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

cd man1
uncompress | tar -xvf -
diff . /usr/man/man1
```

Figure 2.29 UCMD Manager for Windows - Directory Restore Script Example

Figure 2.30, below, illustrates the command to execute the script file.

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

Figure 2.30 UCMD Manager for Windows - Directory Backup Example

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

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

Command Line Options

The command line options used in this example are:

Option	Description
<code>-script</code>	File name of a script file. The script file is sent to the remote system for execution.
<code>-host</code>	Directs the command to a computer with a host name of dal1as .
<code>-userid</code>	Remote user ID with which to execute the command.
<code>-pwd</code>	Password for the user ID.
<code>-stdin</code>	Starts the stdin option list. All options read afterwards are applied to the stdin file. The first option not recognized as a standard file option terminates the stdin option list.
<code>-mode</code>	Transfer mode for the stdout file: binary . The data is not translated.

Components

[Universal Command Manager for Windows](#)

2.3.20 Provide Network Status of Remote UNIX from Windows

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

Figure 2.31, below, illustrates the script file `myscript`.

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

```

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

```

Figure 2.31 UCMD Manager for Windows - System Status Script File Example

Figure 2.32, below, illustrates the command to execute the script file.

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

Figure 2.32 UCMD Manager for Windows - System Status Example

The report is written to the standard out of UCMD.

Command Line Options

The command line options used in this example are:

Option	Description
<code>-script</code>	File name of a script file. The script file is sent to the remote system for execution.
<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 command.
<code>-pwd</code>	Password for the user ID.

Components

Universal Command Manager for Windows

2.3.21 Redirect Standard Out and Standard Error to Windows

Figure 2.33, below, illustrates how to redirect the standard output and error of the 'DIR' command to a file on the initiating system.

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

Figure 2.33 UCMD Manager for Windows - Redirect Standard Out and Standard Error

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

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

Command Line Options

The command line options used in this example are:

Option	Description
<code>-cmd</code>	Remote command <code>ls</code> to execute.
<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 command.
<code>-pwd</code>	Password for the user ID.

Components

Universal Command Manager for Windows

2.3.22 Spawn Background Process with nohup: UNIX

The UCMD Manager job will not end until the remote process ends and all standard files are closed. If a script starts a process that will not end until stopped, the manager process will not end until the process is stopped. In order to start the process and continue processing without waiting for the process to be stopped and the close of the standard files, start the process in the background using the **NOHUP** command and redirect standard out and error to `/dev/null`.

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

Figure 2.34 UCMD Manager for Windows - Spawn Background Process with nohup: UNIX

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

Command Line Options

The command line options used in this example are:

Option	Description
<code>-cmd</code>	Remote command 1s to execute.
<code>-host</code>	Directs the command to a computer with a host name of dallas .
<code>-userid</code>	Remote user ID with which to execute the command.
<code>-pwd</code>	Password for the user ID.

Components

[Universal Command Manager for Windows](#)

2.3.23 Redirect Standard Input from Initiating System to Windows

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

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

Figure 2.35 UCMD Manager for Windows - Redirect Standard Input from Initiating System

The command is sent to a remote system named `da11as` for execution. The output of the script is redirected back to the Universal Command process's standard out and standard error. Standard input is read from file `input.file` on the initiating system. The process will authenticate and run under the authority of userid `joe`.

Command Line Options

The command line options used in this example are:

Option	Description
<code>-script</code>	File from which to read a script file. The script file is sent to the remote system for execution.
<code>-host</code>	Directs the command to a computer with a host name of <code>da11as</code> .
<code>-userid</code>	Remote user ID with which to execute the command.
<code>-pwd</code>	Password for the user ID.

Components

Universal Command Manager for Windows

2.3.24 Redirect Standard Input from /dev/null to Windows

If the command `ucmd` is executed as a background job (using the `&` operator), it will receive the SIGTTIN signal when `ucmd` tries to read from standard input. Background jobs cannot read their standard input from the terminal since the foreground job (or the shell) has it allocated. The `ucmd` job is stopped until it is brought to the foreground. To run an `ucmd` job that does not require terminal input in the background, redirect its standard input from `/dev/null`.

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

Figure 2.36 UCMD Manager for Windows - Redirect Standard Input from /dev/null

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

Command Line Options

The command line options used in this example are:

Option	Description
<code>-script</code>	File from which to read a script file. The script file is sent to the remote system for execution.
<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 command.
<code>-pwd</code>	Password for the user ID.

Components

Universal Command Manager for Windows

2.3.25 Authenticate from Windows Using Encrypted Parameters

Figure 2.37, below, illustrates how to read UCMD parameters from an encrypted file. The encrypted file was previously created using the Universal Encrypt utility with the parameters `-userid` and `-pwd`.

```
ucmd -script script.file -host dallas -encryptedfile encrypted.file
```

Figure 2.37 UCMD Manager for Windows - Authentication Parameters from Encrypted File

The command is sent to a remote system named `dallas` for execution. The output of the script execution is redirected back to the UCMD process. Additional command line options are read from the encrypted file `encrypted.file`.

Command Line Options

The command line options used in this example are:

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

Components

Universal Command Manager for Windows

2.3.26 Universal Submit Job from Windows to IBM i

Figure 2.38, below, illustrates the issuing of a command to the remote IBM i as a parameter of the USBMJOB.

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

Figure 2.38 Universal Submit Job – Windows to IBM i

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

[Universal Command Manager for Windows](#)

[Universal Submit Job](#)

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

Figure 2.39, below, illustrates the use of Universal Return Code to exit with the return code of a command in the middle of the script.

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

A user variable called RC is set to the value of the **ERRORLEVEL** of the previous command. The last line of the script then uses that value as the URC value to set the return code of the script equal to the return code of the **backup.exe** program.

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

Figure 2.39 Universal Return Code - Universal Command Manager for z/OS
Executing URC within a Script

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

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

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

Components

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

[Universal Return Code](#)

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

Figure 2.40, below, provides an example that builds onto the Figure 2.39 example by adding a step that executes the Universal Message Translator (UMET) utility.

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

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

Figure 2.40 Universal Command Manager for z/OS Executing URC and UMET within a Script

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

Components

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

[Universal Return Code](#)

[Universal Message Translator](#)

2.3.29 Provide Network Status of Remote Windows from UNIX

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

Figure 2.41, below, illustrates the script file, `myscript`.

```

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

```

Figure 2.41 UCMD Manager for UNIX - Network Status Script File Example

Figure 2.42, below, illustrates the command to execute the script file.

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

Figure 2.42 UCMD Manager for UNIX - System Status Example

The report is written to the standard out of UCMD.

Command Line Options

The command line options used in this example are:

Option	Description
<code>-script</code>	File name of a script file. The script file is sent to the remote system for execution.
<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 command.
<code>-pwd</code>	Password for the user ID.

Components

Universal Command Manager for UNIX

2.3.30 Redirect Standard Out and Standard Error to UNIX

Figure 2.43, below, illustrates how to redirect the standard output and error of the DIR command to a file on the initiating system.

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

Figure 2.43 UCMD Manager for UNIX - Redirect Standard Out and Standard Error

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

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

Command Line Options

The command line options used in this example are:

Option	Description
<code>-cmd</code>	Remote command <code>ls</code> to execute.
<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 command.
<code>-pwd</code>	Password for the user ID.

Components

Universal Command Manager for UNIX

2.3.31 Redirect Standard Input from Initiating System to UNIX

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

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

Figure 2.44 UCMD Manager for UNIX - Redirect Standard Input from Initiating System

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

Command Line Options

The command line options used in this example are:

Option	Description
<code>-script</code>	File from which to read a script file. The script file is sent to the remote system for execution.
<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 command.
<code>-pwd</code>	Password for the user ID.

Components

Universal Command Manager for UNIX

2.3.32 Redirect Standard Input from /dev/null to UNIX

If the command `ucmd` is executed as a background job (using the `&` operator), it will receive the `SIGTTIN` signal when UCMD tries to read from standard input. Background jobs cannot read their standard input from the terminal since the foreground job (or the shell) has it allocated. The UCMD job is stopped until it is brought to the foreground. To run a UCMD job that does not require terminal input in the background, redirect its standard input from `/dev/null`.

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

Figure 2.45 UCMD Manager for UNIX - Redirect Standard Input from /dev/null

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

Command Line Options

The command line options used in this example are:

Option	Description
<code>-script</code>	File from which to read a script file. The script file is sent to the remote system for execution.
<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 command.
<code>-pwd</code>	Password for the user ID.

Components

Universal Command Manager for UNIX

2.3.33 Universal Submit Job from UNIX to IBM i

Figure 2.46, below, illustrates the issuing of a command to the remote IBM i as a parameter of the USBMJOB.

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

Figure 2.46 Universal Submit Job – UNIX to IBM i

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

[Universal Command Manager for UNIX](#)

[Universal Submit Job](#)

2.3.34 Provide Network Status of Remote Windows from IBM i

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

Figure 2.47, below, illustrates the script file, **MYSCRIPT**.

```

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

```

Figure 2.47 UCMD Manager for IBM i - Network Status Script File

Figure 2.48, below, illustrates the command to execute the script file.

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

Figure 2.48 UCMD Manager for IBM i - Network Status Script File Command

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

Command Line Options

The command line options used in this example are:

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

Components

Universal Command Manager for IBM i

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

Figure 2.49, below, illustrates the execution of a network status script on a remote Windows server.

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

Figure 2.49 UCMD Manager for IBM i - Command Coded as a Script

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

Command Line Options

The command line options used in this example are:

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

Components

Universal Command Manager for IBM i

2.3.36 Display Library with Manager Fault Tolerance Active Using USBMJOB

Figure 2.50, below, illustrates the use of an IBM i command on a remote system with spooling enabled.

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

```
Windows System  
ucmd -cmd "USBMJOB CMD(dspLib qgp1)" -userid userId -pwd password  
      -host sysName -cmdid NTSysTest <NUL 1>out400.txt 2>err400.txt  
  
UNIX System  
ucmd -cmd "USBMJOB CMD(dspLib qgp1)" -userid userId -pwd userPW  
      -host sysName -cmdid UNIX00 1>out400.txt 2>err400.txt
```

Figure 2.50 UCMD Manager for IBM i - Display Library

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

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

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

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

Command Line Options

The command line options used in this example are:

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

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

Components

[Universal Command Manager for IBM i:](#)

2.3.37 Universal Submit Job from z/OS to IBM i

Figure 2.51, below, illustrates the issuing of a command to the remote IBM i as a parameter of the USBMJOB.

```
//S1 EXEC UCMDPRC
//UNVOUT DD SYSOUT=*
//UNVERR DD SYSOUT=*
//SCRIPT DD *
  ADDLIB lib(UNVPRD420)
  UNVPRD420/USBMJOB CMD(dsp1ib tuser1)
//SYSIN DD *
  -script SCRIPT
  -host as400 -userid tuser1 -pwd tuser1
/*
```

Figure 2.51 Universal Submit Job - z/OS to IBM i

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

The script runs with the authority of UserID **tuser1**. The first line of the script adds the library **UNVPRD420** to the library concatenation of user **tuser1**. The second line executes the command **dsp1ib tuser1** with the USBMJOB utility.

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

SYSIN Options

The SYSIN options used in this example are:

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

USBMJOB Options

The USBMJOB option used in this example is:

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

Components

[Universal Command Manager for IBM i](#)

[Universal Submit Job](#)

2.3.38 Provide Network Status of Remote Windows from HP NonStop

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

Figure 2.52, below, illustrates the script file, `myscript`.

```

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

```

Figure 2.52 UCMD Manager for HP NonStop - Network Status Script File

Figure 2.53, below, illustrates the command to execute the script file.

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

Figure 2.53 UCMD Manager for HP NonStop - System Status Example

The report is written to the standard out of UCMD.

Command Line Options

The command line options used in this example are:

Option	Description
<code>-script</code>	File name of a script file. The script file is sent to the remote system for execution.
<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 command.
<code>-pwd</code>	Password for the user ID.

Components

Universal Command Manager for HP NonStop

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

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

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

Figure 2.54 UCMD Manager for HP NonStop - Command Coded as a Script

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

Command Line Options

The command line options used in this example are:

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

Components

[Universal Command Manager for HP NonStop](#)

Remote Execution for SAP Systems

3.1 Overview

This chapter provides information on the Remote Processing for SAP Systems feature of Indesca.

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

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

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

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

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

3.1.1 Work Requests

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

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

3.2 Mass Activities Support in Universal Connector

Universal Connector supports the submission, starting, and monitoring of mass activities on the SAP system.

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

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

The basic process flow in working with mass activities is:

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

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

3.2.1 Initiating Mass Activities

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

For more information on submitting and starting jobs with Universal Connector, see the `SUBMIT`, `START`, and `RUN` commands in Chapter 2 [Universal Connector for z/OS](#) and/or Chapter 3 [Universal Connector for UNIX](#) of the [Universal Connector Reference Guide](#).

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

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

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

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

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

For additional information on working with variants, refer to the `SUBMIT VARIANT` and `MODIFY VARIANT` commands in Chapter 2 [Universal Connector for z/OS](#) and/or Chapter 3 [Universal Connector for UNIX](#) of the [Universal Connector Reference Guide](#).

3.2.2 Monitoring Mass Activities

Regardless of which program is used to initiate a mass activity, Universal Connector follows the same procedure for monitoring the process to completion. The `MASS_ACTIVITY_WAIT` command is used to instruct Universal Connector that it should perform this monitoring function (refer to the `MASS_ACTIVITY_WAIT` option in [Chapter 2 Universal Connector for z/OS](#) and/or [Chapter 3 Universal Connector for UNIX](#)).

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

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

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

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

3.2.3 Working with Parameter Records

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

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

For more information, refer to the `SUBMIT`, `START`, `RUN`, and `WAIT` commands in [Chapter 2 Universal Connector for z/OS](#) and/or [Chapter 3 Universal Connector for UNIX](#) of the [Universal Connector Reference Guide](#).

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

For additional information on working with variants, refer to the `SUBMIT VARIANT` and `MODIFY VARIANT` commands in [Chapter 2 Universal Connector for z/OS](#) and/or [Chapter 3 Universal Connector for UNIX](#) of the [Universal Connector Reference Guide](#).

3.3 Mass Activities Support Examples

This section contains examples demonstrating the use of Universal Connector mass activities support.

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

z/OS

- [Universal Connector Mass Activity Example 1](#)

3.3.1 Universal Connector Mass Activity Example 1

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

Step 1

Via SAP GUI, create a template parameter set for a mass run:

1. In the SAP Front end GUI, enter transaction **FPCC0002** to bring up a dialog for the Create Account Statements application.
2. In the Run Identification section, enter values for Date ID and Identification. These values will be used to uniquely identify the parameter set that you create in this step.
For example:
 - Date ID: 13.07.2010
 - Identification: SBX1
3. Set up the rest of the application run parameters to meet your needs. These additional settings are not important to the concepts of this example.
4. Select Program Run->Save (or <Ctrl+S>) to save the parameter set.

Step 2

Create variant for RFKK_MA_SCHEDULER.

1. Create a variant for ABAP program RFKK_MA_SCHEDULER that will use the parameter set created in **Step 1**. This can be accomplished by running the sample job illustrated in [Figure 3.1](#) and [Figure 3.2](#), below.
The procedure for executing this JCL is illustrated in [Figure 3.4 USPPRC – JCL Procedure](#).

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

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

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

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

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

Figure 3.1 USPVRMS – JCL (1 of 2)


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

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

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

/* WF_OKEY */
SELNAME      = "WF_OKEY"
  KIND       = "P"
  LOW        = "";

/* WF_WITEM */
;
SELNAME      = "WF_WITEM"
  KIND       = "P"
  LOW        = "";

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

Figure 3.2 USPVRMS – JCL (2 of 2)

Step 3

Via Universal Connector, run ABAP program `RFKK_MA_SCHEDULER`.

1. Submit a Universal Connector job to initiate, monitor, and return output from the mass activity.

The sample job illustrated in [Figure 3.3](#), below, will accomplish this using the variant created in **Step 2**.

The procedure for executing this JCL is illustrated in [Figure 3.4 USPPRC – JCL Procedure](#).

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

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

Figure 3.3 USPJRMS – JCL

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

Figure 3.4 USPPRC – JCL Procedure

Components

Universal Connector for z/OS

3.4 Batch Input Monitoring in Universal Connector

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

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

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

3.4.1 Batch Input Monitoring Process

A basic overview of the Universal Connector batch input monitoring process follows:

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

3.4.2 Batch Input Monitoring Requirements

SAP System

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

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

SAP Batch Input Sessions

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

Universal Connector

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

Universal Connector uses the spoolist generated by **RSBDCSUB** to extract session processing information. The format of this report depends on the language of the job step. There are three Universal Connector parameters that must be set up for the language being used. See Sections [2.3.15 BDCWAIT Command \(z/OS\)](#) and [3.3.15 BDCWAIT Command \(UNIX\)](#) in the [Universal Connector Reference Guide](#) for details. By default, these parameters are set up to work with the English language.

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

The Universal Connector command line parameter **-bdcwait** is used to initiate the batch input monitoring process. For details on this parameter, see Sections [2.3.15 BDCWAIT Command \(z/OS\)](#) and [3.3.15 BDCWAIT Command \(UNIX\)](#) in the [Universal Connector Reference Guide](#).

3.4.3 Batch Input Monitoring Parameters

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

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

See Sections [2.3.15 BDCWAIT Command \(z/OS\)](#) and [3.3.15 BDCWAIT Command \(UNIX\)](#) in the [Universal Connector Reference Guide](#) for details concerning the use of these parameters.

3.5 Batch Input Monitoring Examples

This section contains examples demonstrating the use of Universal Connector batch input monitoring.

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

z/OS

[Batch Input Processing Example](#)

3.5.1 Batch Input Processing Example

This example illustrates batch input processing for z/OS.

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

Figure 3.5 Batch Input Processing (1 of 2)


```
/* Session */
SELNAME      = "MAPPE"
  KIND       = "P"
  SIGN       = ""
  OPTION     = ""
  LOW        = "SBX20100720"
  HIGH       = "";
//SYSIN      DD *
  -dest      CF5
  -client    800
  -userid    sapuid
  -pwd       sappwd
  -modify    VARDEF
/*
//*****
/* Run ABAP program RSBDCSUB to perform Batch Input processing
/* using the variant that was modified in step 1.
/*
/* NOTE: This job requires that a variant SBX1 exists for ABAP
/* program RSBDCSUB.
/*
//*****
//STEP2      EXEC USPPRC
//JOBDEF     DD *
/* Job Header statement. */
JOBNAME = "RSBDCSUB";
/* ABAP Step statement. */
ABAP_STEP      = "STEP 1"
  ABAP_PROGRAM_NAME = "RSBDCSUB_SBX1"
  VARIANT_NAME    = "SBX1";
//SYSIN      DD *
  -dest      CF5
  -client    800
  -userid    sapuid
  -pwd       sappwd
  -sub       JOBDEF
  -start
  -bdcwait
/*
```

Figure 3.6 Batch Input Processing (2 of 2)

SYSIN Options

SYSIN options used in this example are:

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

Components

Universal Connector for z/OS

3.6 Remote Execution for SAP Systems Examples

This section contains examples demonstrating the use of Universal Connector to define SAP jobs.

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

z/OS and UNIX

[Define Job, Run Job, Get Output, and Purge Job](#)

z/OS

[Submitting a Job to an SAP System Using a Pre-existing SAP Job as a Template – z/OS](#)

[Submitting a Job to an SAP System Using a Universal Connector Job Definition File – z/OS](#)

[Running a Job on an SAP System Using a Pre-existing SAP Job – z/OS](#)

[Running a Job on an SAP System Using a Universal Connector Job Definition File – z/OS](#)

[Running an SAP Job on a Specific SAP Server – z/OS](#)

[Variant Substitution – z/OS](#)

[Creating a USAP Variant Definition Using the Universal Connector GENERATE VARDEF Command – z/OS](#)

[Creating a Universal Connector Job Definition Using the GENERATE JOBDEF Command – z/OS](#)

UNIX

[Submitting a Job to an SAP System Using a Pre-existing SAP Job as a Template – UNIX](#)

[Submitting a Job to an SAP System Using a Universal Connector Job Definition File – UNIX](#)

[Running a Job on an SAP System Using a Pre-existing SAP Job – UNIX](#)

[Running a Job on an SAP System Using a Universal Connector Job Definition File – UNIX](#)

[Running an SAP Job on a Specific SAP Server – UNIX](#)

[Variant Substitution – UNIX](#)

[Creating a USAP Variant Definition Using the USAP GENERATE VARDEF Command – UNIX](#)

[Creating a USAP Job Definition Using the USAP GENERATE JOBDEF Command – UNIX](#)

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

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

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

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

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

Command Options

The command options used are:

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

Components

[Universal Connector for z/OS](#)

[Universal Connector for UNIX](#)

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

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

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

Figure 3.7 Submitting a Job to an SAP System Using a Pre-existing SAP Job as a Template – z/OS

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

SYSIN Options

SYSIN options used in this example are:

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

Components

[Universal Connector for z/OS](#)

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

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

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

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

Figure 3.8 Submitting a Job to an SAP System Using a USAP Job Definition File – z/OS

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

SYSIN Options

SYSIN options used in this example are:

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

Components

[Universal Connector for z/OS](#)

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

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

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

Figure 3.9 Running a Job on an SAP System Using a Pre-existing SAP Job – z/OS

SYSIN Options

SYSIN options used in this example are:

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

Components

Universal Connector for z/OS

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

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

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

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

Figure 3.10 Running a Job on an SAP System Using a Universal Connector Job Definition File – z/OS

SYSIN Options

SYSIN options used in this example are:

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

Components

Universal Connector for z/OS

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

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

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

/* ABAP Step statement. */
ABAP_STEP          = "STEP 1"
  ABAP_PROGRAM_NAME = "BTCSPool";
//SYSIN DD *
  -dest          CF5
  -client        800
  -userid        sapuid
  -pwd           sappwd
  -sub           JOBDEF
  -start
  -targetserver  pdf2643
  -wait
  -joblog        yes
  -spoolist      yes
/*
```

Figure 3.11 Running an SAP Job on a Specific SAP Server – z/OS

SYSIN Options

SYSIN options used in this example are:

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

Components

Universal Connector for z/OS

3.6.7 Variant Substitution – z/OS

This example demonstrates the use of variant substitution.

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

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

Figure 3.12 Variant Substitution – z/OS (1 of 3)


```

//          JCLLIB ORDER=#SHLQ.UNV.SUNVSAMP
//*
//*****
/* Modify variant 'SBT1' for ABAP program RSUSR002
//*****
//STEP1     EXEC USPPRC
//VARDEF   DD  *
/* Variant Header statement. */
VARIANT_NAME = "SBT1"
  REPORT     = "RSUSR002";
/* User */
SELNAME     = "USER"
  KIND       = "S"
  SIGN       = "I"
  OPTION     = "CP"
  LOW        = "STONEBRANCH"
  HIGH       = "";
//SYSIN    DD  *
  -dest      CF5
  -client    800
  -userid    sapuid
  -pwd       sappwd
  -modify    VARDEF
/*
//*****
/* Modify variant 'SBT2' for ABAP program RSUSR002
//*****
//STEP2     EXEC USPPRC
//VARDEF   DD  *
/* Variant Header statement. */
VARIANT_NAME = "SBT2"
  REPORT     = "RSUSR002";
/* User */
SELNAME     = "USER"
  KIND       = "S"
  SIGN       = "I"
  OPTION     = "CP"
  LOW        = "STONEBRANCH1"
  HIGH       = "";

```

Figure 3.13 Variant Substitution – z/OS (2 of 3)

```
//SYSIN DD *
  -dest          CF5
  -client        800
  -userid        sapuid
  -pwd           sappwd
  -modify        VARDEF
/*
//*****
/** Run SAP job using a pre-defined SAP job as a template and
/** perform variant substitution.
/**
/** NOTE: This job assumes (and requires) that a job already
/** exists on an the SAP system with:
/** Job Name: VARSBST1
/** Job ID: 12345678
/**
/** The pre-defined job must have ABAP program RSUSR002 defined in
/** step one and step two.
//*****
//STEP3 EXEC USPPRC
//SYSIN DD *
  -dest          CF5
  -client        800
  -userid        sapuid
  -pwd           sappwd
  -run
  -jobname       VARSBST1
  -jobid         12345678
  -target_variant 1,SBT1;2,SBT2
/*
```

Figure 3.14 Variant Substitution – z/OS (3 of 3)

SYSIN Options

SYSIN options used in this example are:

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

Components

Universal Connector for z/OS

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

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

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

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

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

Figure 3.15 Creating a USAP Variant Definition Using the USAP GENERATE VARDEF Command – z/OS

SYSIN Options

SYSIN options used in this example are:

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

Components

Universal Connector for z/OS

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

SAP jobs offer many configuration options. Creating Universal Connector job definitions that utilize many configuration options by hand can be tedious and time consuming.

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

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

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

Figure 3.16 Creating a Universal Connector Job Definition Using the GENERATE JOBDEF Command – z/OS

SYSIN Options

SYSIN options used in this example are:

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

Components

[Universal Connector for z/OS](#)

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

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

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

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

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

Figure 3.17, below, illustrates the command to submit the job.

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

Figure 3.17 Submitting a Job to an SAP System Using a Pre-existing SAP Job as a Template – UNIX

Command Line Options

Command line options used in this example are:

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

Components

Universal Connector for UNIX

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

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

Figure 3.18, below illustrates the job definition file.

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

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

Figure 3.18 Submitting a Job to an SAP System Using a Universal Connector Job Definition File – UNIX

Figure 3.19, below, illustrates the command to submit the job.

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

Figure 3.19 Submitting a Job to an SAP System - Command to Submit Job – UNIX

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

Command Line Options

Command line options used in this example are:

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

Components

Universal Connector for UNIX

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

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

Executing this example will:

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

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

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

Figure 3.20, below, illustrates the command to run the job.

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

Figure 3.20 Running a Job on an SAP System Using a Pre-existing SAP Job – UNIX

Command Line Options

Command line options used in this example are:

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

Components

Universal Connector for UNIX

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

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

Executing this sample will:

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

Figure 3.21, below illustrates the job definition file.

```
/* Job Header statement. */  
JOBNAME = "USPRUN";  
  
/* ABAP Step statement. */  
ABAP_STEP          = "STEP 1"  
ABAP_PROGRAM_NAME = "BTCSPool";
```

Figure 3.21 Running a Job on an SAP System - USAP Job Definition File

Figure 3.22, below, illustrates the command to run the job.

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

Figure 3.22 Running a Job on an SAP System - Command to Run Job

Command Line Options

Command line options used in this example are:

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

Components

Universal Connector for UNIX

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

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

Executing this example will:

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

Figure 3.23, below illustrates the job definition file.

```
/* Job Header statement. */  
JOBNAME = "USPRUN3";  
  
/* ABAP Step statement. */  
ABAP_STEP          = "STEP 1"  
  ABAP_PROGRAM_NAME = "BTCSPool";
```

Figure 3.23 Running a Job on a Specific SAP Server - USAP Job Definition File

Figure 3.24, below, illustrates the command to run the job.

```
usap -dest CF5 -client 800 -userid sapuid -pwd sappwd  
  -sub jobdefFile -start -targetserver pddf2643 -wait  
  -joblog yes -spoollist yes
```

Figure 3.24 Running a Job on a Specific SAP Server - Command to Run Job

Command Line Options

Command line options used in this example are:

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

Components

Universal Connector for UNIX

3.6.15 Variant Substitution – UNIX

This example demonstrates the use of variant substitution.

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

This example is comprised of three steps:

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

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

Executing this example will:

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

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

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

Step One

This step modifies SAP variant **SBT1**.

[Figure 3.25](#), below, illustrates the variant definition file for variant **SBT1**.

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

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

Figure 3.25 Variant Substitution for Variant SBT1- Variant Definition File

[Figure 3.26](#), below, illustrates the command line to modify variant **SBT1**.

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

Figure 3.26 Variant Substitution for Variant SBT1 - Command Line

Step Two

This step modifies SAP variant **SBT2**.

[Figure 3.27](#), below, illustrates the variant definition file for variant **SBT2**.

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

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

Figure 3.27 Variant Substitution for Variant SBT2- Variant Definition File

[Figure 3.28](#), below, illustrates the command line to modify variant **SBT2**.

```

usap -dest CF5 -client 800 -userid sapuid -pwd sappwd
      -modify vardefFile2

```

Figure 3.28 Variant Substitution for Variant SBT2 - Command Line

Step Three

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

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

Figure 3.29, below, illustrates the variant definition file for variant **SBT2**.

```

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

```

Figure 3.29 Command Line to Run a New Job using Variant Substitution

Command Line Options

Command line options used in this example are:

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

Components

Universal Connector for UNIX

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

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

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

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

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

Figure 3.30, below, illustrates the command used to generate a Universal Connector variant definition.

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

Figure 3.30 Creating a USAP Variant Definition Using the USAP GENERATE VARDEF Command

Command Line Options

Command line options used in this example are:

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

Components

Universal Connector for UNIX

3.6.17 Creating a USAP Job Definition Using the USAP GENERATE JOBDEF Command – UNIX

SAP jobs offer many configuration options. Creating Universal Connector job definitions that utilize many configuration options by hand can be tedious and time consuming.

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

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

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

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

Figure 3.31, below, illustrates the command used to generate a Universal Connector job definition.

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

Figure 3.31 Creating a USAP Job Definition Using the USAP GENERATE JOBDEF Command

Command Line Options

Command line options used in this example are:

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

Components

Universal Connector for UNIX

Web Services Execution

4.1 Overview

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

4.2 Outbound Implementation

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

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

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

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

Indesca enables you to:

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

(See Section [4.4 Web Services - Outbound Examples](#).)

4.3 Inbound Implementation

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

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

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

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

(See Section [4.5 Web Services - Inbound Examples](#).)

4.4 Web Services - Outbound Examples

This section provides examples for the Web Services Execution feature of Indesca.

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

Windows and UNIX

[Basic Structure of Using Indesca to Publish to a SOA Workload](#)

[Message Payload for SOAP](#)

[Logging Configuration](#)

UNIX

[Outbound SOAP Implementation](#)

4.4.1 Basic Structure of Using Indesca to Publish to a SOA Workload

[Figure 4.1](#), below, illustrates the basic structure of using Indesca to publish to a SOA workload.

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

Figure 4.1 Basic Structure of Using Indesca to Publish to a SOA Workload

Command Line Options

The command line options used in this example are:

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

The contents of the file `options.txt` define the type of workload being published to (see [Example Workloads](#)).

Components

[Universal Command](#)

[Universal Command Agent for SOA](#)

Example Workloads

Below are examples of various workloads.

JMS ActiveMQ Workload

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

Figure 4.2 JMS ActiveMQ Workload

JMS Websphere Workload

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

Figure 4.3 JMS Websphere Workload

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

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

XD Workload

```
-xcmd SUBMIT
-xcmdid XDJOB1
-servicepassword xdservicepass
-protocol XDSOAP
-serviceurl http://soatest2:9080/LongRunningJobSchedulerwebsvcRouter/services/JobScheduler
-serviceusername xdusername
-timeoutsec 120
-mep REQUEST
```

Figure 4.4 XD Workload

MQ Series Workload

```
-mqqueuemanagename MyQueueManager
-mqqueueename UpsQaQueue
-mqhost soatest2
-mqchannel UpsQaChannel
-protocol mq
-timeoutsec 120
-mep publish
```

Figure 4.5 MQ Series Workload

4.4.2 Message Payload for SOAP

Figure 4.6, below, illustrates an example of a basic message payload for SOAP.

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

Figure 4.6 Message Payload - SOAP

The first line contains:

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

The second line contains:

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

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

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

SOAP Response

Figure 4.7, below, illustrates the SOAP response that the ValidateZip operation returns.

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

Figure 4.7 Message Payload Response - SOAP

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

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

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

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

Components

[Universal Command Agent for SOA](#)

4.4.3 Logging Configuration

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

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

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

The logging levels supported by the logging implementation are:

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

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

Components

[Universal Command Agent for SOA](#)

uac_log4jConfiguration.xml Example

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

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

Figure 4.8 uac_log4jConfiguration.xml Example

uai_log4jConfiguration.xml Example

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

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

Figure 4.9 uai_log4jConfiguration Example

4.4.4 Outbound SOAP Implementation

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

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

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

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

The following JCL will initiate the outbound SOAP request.

```

//TZE025R2 JOB (TEST,CC0KG150000), 'WINDOWS',                                JOB08030
//          CLASS=S,
//          MSGCLASS=R
//*
// JCLLIB ORDER=TEST.SYS5.UNV.SUNVSAMP
//* *****
//* * Sample SOA Communication for R1
//* * *****
//* * STEPS - FUNCTION
//* * -----
//* * SYSIN - Target destination for process / LINUX
//* * INPUT - Universal Command Options to execute SOAP
//* * UNVIN - PAYLOAD being passed to server
//* *****
//STEP1    EXEC UCMDPRC
//LOGIN    DD  DISP=SHR,DSN=ZE025.PROD.INDESCA(IDNPSWD)
//SYSIN    DD  DISP=SHR,DSN=ABC.CONTROL.UPARMLIB(HOSTPARM)
//INPUT    DD  DISP=SHR,DSN=ABC.CONTROL.UPARMLIB(SOAPCALL)
//UNVIN    DD  DISP=SHR,DSN=ABC.CC030210.PMS002.STGXML.START

```

Figure 4.10 Outbound SOAP Request - JCL

This JCL executes the Universal Command JCL procedure.

The DD Statements contain the following:

LOGIN DD

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

SYSIN DD

Universal Command runtime parameters:

- **-HOST**
DNS name or IP address of the Linux Server running Universal Command Agent for SOA.
- **-ENCRYPTEDFILE**
Specified the DD name that will contain the encrypted password file.
- **-SCRIPT**
Specifies the DD name that will contain the Universal Command Agent for SOA runtime parameters that are passed to the Universal Command for Agent SOA.
- **-SCRIPT_TYPE**
The value **SERVICE** tells Universal Command that this is a SOA process.

-HOST	deveis01
-ENCRYPTEDFILE	LOGIN
-SCRIPT	INPUT
-SCRIPT_TYPE	SERVICE

Figure 4.11 Outbound SOAP Request - SYSIN DD Contents

INPUT DD

Universal Command Agent for SOA runtime parameters:

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

```

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

```

Figure 4.12 Outbound SOAP Request - SYSIN DD Contents

UNVIN DD

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

```

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

```

Figure 4.13 Outbound SOAP Request - SYSIN DD Contents

Components

[Universal Command](#)

[Universal Command Agent for SOA](#)

[Universal Encrypt](#)

4.5 Web Services - Inbound Examples

This section provides examples of inbound implementation of the Web Services Execution feature of Indesca.

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

Windows and UNIX

[Inbound JMS Examples](#)

[Inbound SOAP Implementation](#)

4.5.1 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 4.14, 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 4.14 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 4.15, 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 4.15 Inbound JMS - Attachment to an Apache ActiveMQ Dynamic Topic

Websphere Queue

Figure 4.16, 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 4.16 Inbound JMS - Attachment to an IBM Websphere Queue

MQ Series Queue:

Figure 4.17, 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:MQListener>
      <sb:QueueName>UpsQaQueue</sb:QueueName>
      <sb:Actions>
        <sb:MQFilewriter>
          <sb:Directory>filesystem</sb:Directory>
          <sb:FilenamePattern>MQSeries_Queue_%Seq%.txt</sb:FilenamePattern>
          <sb:StartSequenceNumber>0</sb:StartSequenceNumber>
          <sb:WriteProperties>>false</sb:WriteProperties>
        </sb:MQFilewriter>
      </sb:Actions>
    </sb:MQListener>
  </sb:Listeners>
</sb:MQConnection>
```

Figure 4.17 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 4.18](#), 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 a system command to move the file.

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

begin_handler
  handler_id "JMS_MESSAGE_HANDLER"
  handler_type CMD
  maxrc 0
  userid username
  pwd user_password
  cmd "move ${origname}.xml ${origname}.found"
end_handler
```

Figure 4.18 Triggering an Event

Components

[Universal Event Monitor](#)

[Universal Event Monitor for SOA](#)

4.5.2 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 4.19 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 4.20 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 4.21, 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 4.21 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 4.22 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 4.23 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 4.24 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](#)

Event Monitoring and File Triggering

5.1 Overview

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

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

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

5.2 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 5.1, below, illustrates the interaction of the various components that make up Universal Event Monitor.

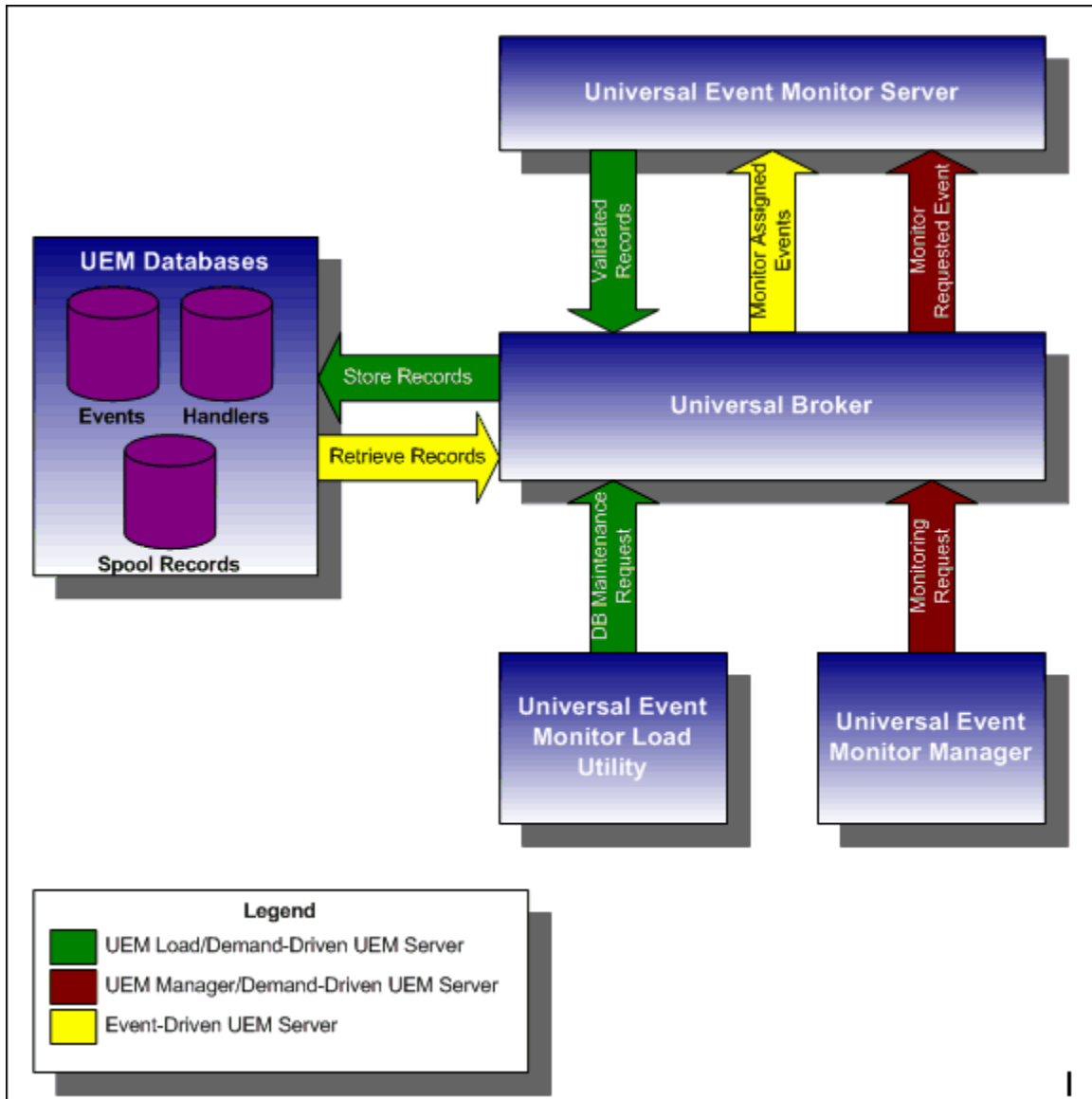


Figure 5.1 High-Level Interaction of UEM Components

5.2.1 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 5.2, below, illustrates the interaction of the Universal Broker and the Universal Event Monitor Server components involved during the execution of UEMLoad.

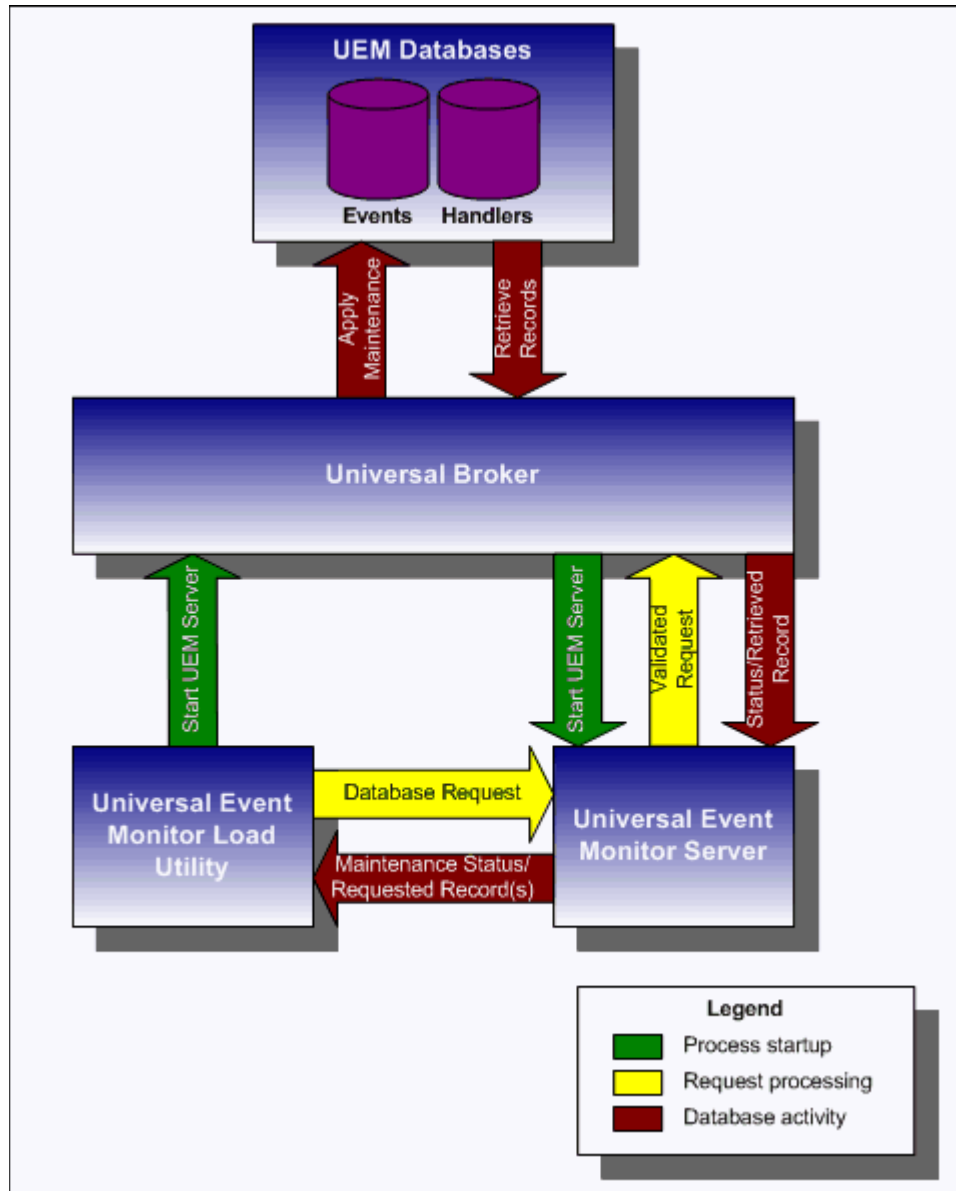


Figure 5.2 UEMLoad Utility Overview

5.2.2 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 5.3, below, illustrates the interaction of the Universal Broker and the Universal Event Monitor components involved when a UEM Manager is executed.

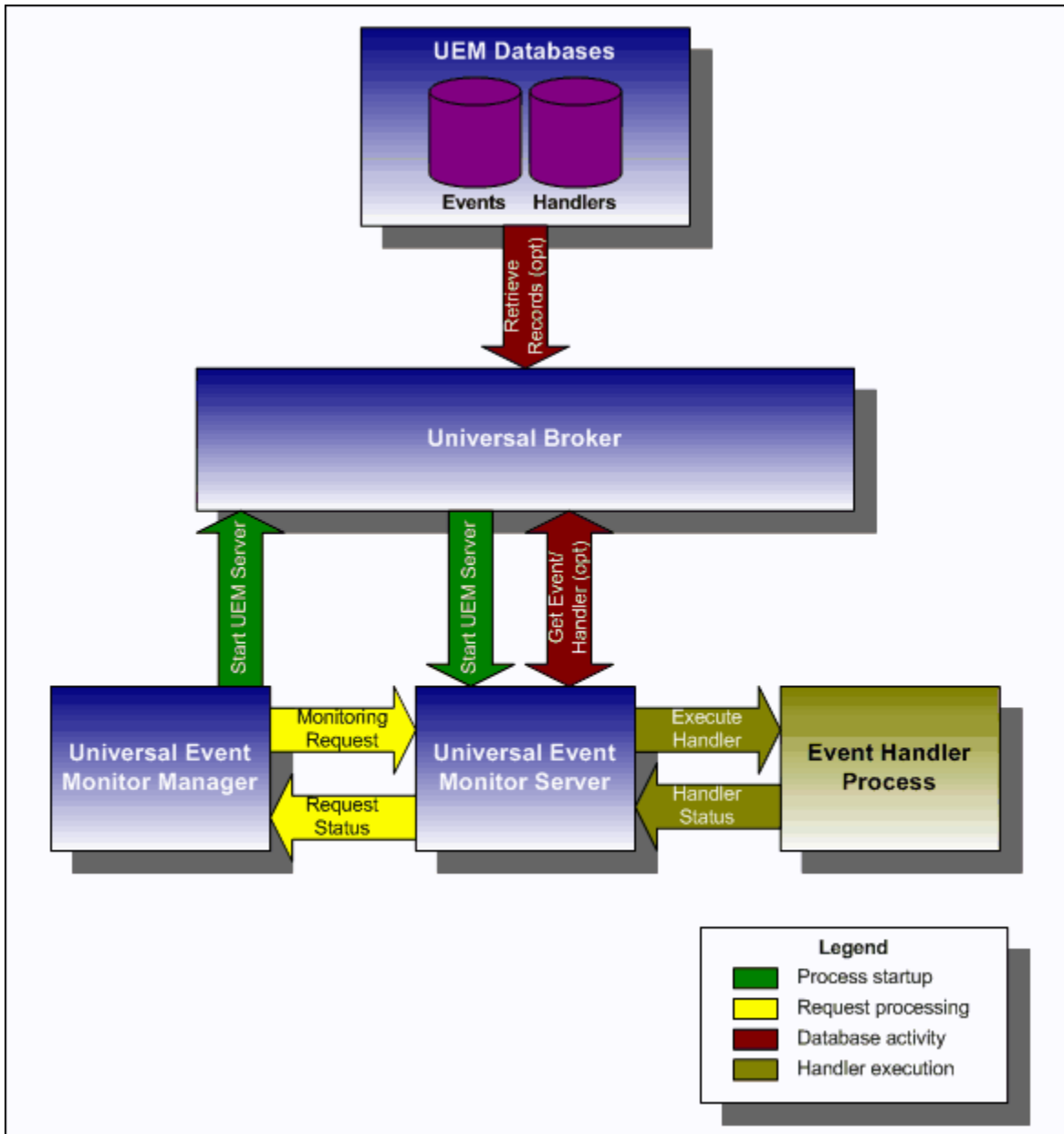


Figure 5.3 UEM Manager Overview

5.2.3 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 5.4, below, illustrates the interaction of the Universal Broker and an event-driven UEM Server.

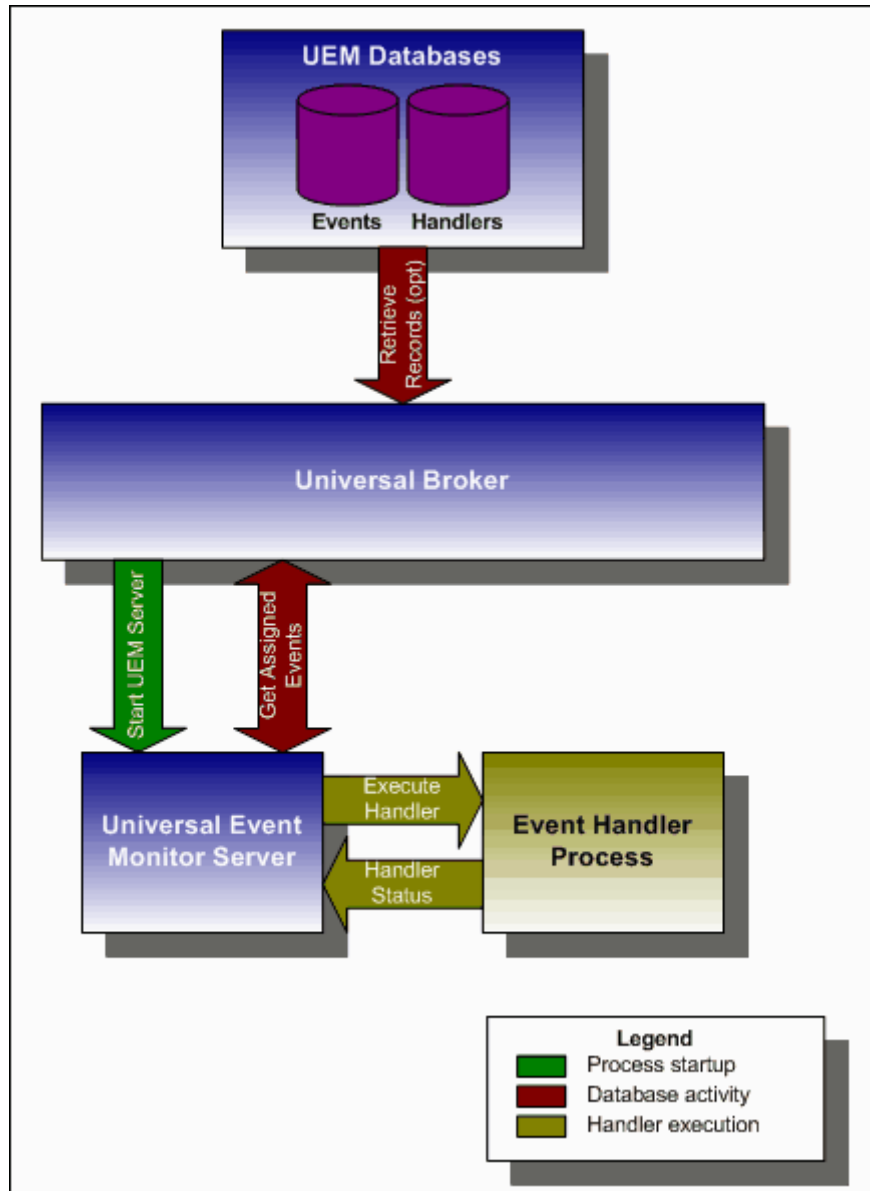


Figure 5.4 UEM Server Overview

5.3 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](#).

5.3.1 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 (`usrm`) 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.

5.4 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 Indesca.

Links to detailed technical information on appropriate Indesca 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](#)

5.4.1 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 5.5 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](#)

5.4.2 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 5.6 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](#)

5.4.3 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 5.7 Using a Stored Event Record

Components

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

[Universal Event Monitor Server](#)

5.4.4 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 5.8 Handling an Event with a Script

Components

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

[Universal Event Monitor Server for Windows](#)

5.4.5 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 5.9 Handling an Expired Event

Components

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

[Universal Event Monitor Server for UNIX](#)

5.4.6 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 5.10 Continuation Character - in z/OS Handler Script

Components

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

5.4.7 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 5.11 Continuation Character + in z/OS Handler Script

Components

Universal Event Monitor Manager for z/OS

5.4.8 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 5.12 Continuation Characters - and + in z/OS Handler Script

Components

Universal Event Monitor Manager for z/OS

5.4.9 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 5.13 Using a Stored Event Handler Record

Components

Universal Event Monitor Manager for Windows

5.4.10 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 5.14 Handling an Event with a Script.

Figure 5.15, 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 5.15 Contents of Sample Script File

Components

[Universal Event Monitor Manager for Windows](#)

[Universal Event Monitor Server for UNIX](#)

5.4.1.1 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` option. 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 5.16 Handling an Expired Event

Components

[Universal Event Monitor Manager for Windows](#)

[Universal Event Monitor Server for UNIX](#)

5.4.12 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 5.17 Adding a single event definition record.

Components

[UEMLoad Utility for Windows](#)

[Universal Event Monitor Server for UNIX](#)

5.4.13 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 5.18 Adding a Single Event Handler Record

Components

[UEMLoad Utility for Windows](#)

[Universal Event Monitor Server for UNIX](#)

[Universal Encrypt](#)

5.4.14 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 5.19 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](#)

5.4.15 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 5.27](#).

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

Figure 5.20 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](#)

5.4.16 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 5.21 List a Single Event Handler Record

Figure 5.22, 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 5.18.)

In this specific instance, the user ID contained in `encrypted.file` (from Figure 5.18) 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 05/25/2010.

Event Handler(s):
=====
Handler ID.....: dirlist
Max Acceptable Return Code.: 0
User ID.....: sparkie
Command.....: ls -al
Last Modified On.....: 05/25/2010 11:32:06 AM
Last Modified By.....: sbuser

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

Figure 5.22 Sample List Output

Components

[UEMLoad Utility for Windows](#)

5.4.17 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 5.23 Using Wildcards to List Records

Components

[UEMLoad Utility for Windows](#)

5.4.18 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 5.27](#).

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 5.24 Add Database Record(s) Using a Definition File

Components

[UEMLoad Utility for Windows](#)

5.4.19 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 5.27](#).

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 5.25 Redirect Definition File from stdin

Components

[UEMLoad Utility for Windows](#)

[Universal Command Manager for Windows](#)

[Universal Event Monitor Server](#)

5.4.20 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 5.27](#).

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 5.26 Redirect Definition File from STDIN (for z/OS)

Components

[UEMLoad Utility for Windows](#)

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

[Universal Event Monitor Server](#)

5.4.21 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 5.27](#), below.

```
# 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 5.27 Definition File Sample - Windows

Components

UEMLoad Utility for Windows

5.4.22 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 5.28 Using a Stored Event Handler Record

Components

[Universal Event Monitor Manager for UNIX](#)

[Universal Event Monitor Server for Windows](#)

5.4.23 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 5.29 Handling an Event with a Script

Figure 5.30, 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 5.30 Contents of Sample Script File

Components

[Universal Event Monitor Manager for UNIX](#)

[Universal Event Monitor Server for Windows](#)

5.4.24 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 the home directory of `uemuser`.

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

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

If the UEM Server does not detect the presence of `uemtest.dat` before the inactive time elapses, it sets the event to the expired state. When this happens, the UEM Server executes the command specified by the `-cmd` option corresponding to the `-expired` option. In this example, UEM executes the `'ls -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 5.31 Handling an Expired Event

Components

[Universal Event Monitor Manager for UNIX](#)

[Universal Event Monitor Server for UNIX](#)

5.4.25 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 5.32 Adding a single event definition record.

Components

[UEMLoad Utility for UNIX](#)

[Universal Event Monitor Server for UNIX](#)

5.4.26 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 5.33 Adding a single event handler record.

Components

[UEMLoad Utility for UNIX](#)

[Universal Event Monitor Server for UNIX](#)

[Universal Encrypt](#)

5.4.27 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 in double quotation marks (that is: `*` or `"*"`, respectively).

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

Figure 5.34 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](#)

5.4.28 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 5.27](#).

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

Figure 5.35 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](#)

5.4.29 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 5.36 List a Single Event Handler Record

Figure 5.37, 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 5.18.)

In this specific instance, the user ID contained in `encrypted.file` (from Figure 5.18) 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 05/25/2010.

Event Handler(s):
=====
Handler ID.....: dirlist
Max Acceptable Return Code.: 0
User ID.....: sparkie
Command.....: ls -al
Last Modified On.....: 05/25/2010 11:32:06 AM
Last Modified By.....: sbuser

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

Figure 5.37 Sample List Output

Components

[UEMLoad Utility for UNIX](#)

5.4.30 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 5.38 Using Wildcards to List Records

Components

[UEMLoad Utility for UNIX](#)

5.4.31 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 5.27](#).

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 5.39 Add Database Record(s) Using a Definition File

Components

[UEMLoad Utility for UNIX](#)

5.4.32 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 5.27](#).

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 5.40 Redirect Definition File from stdin

Components

[Universal Command Manager for UNIX](#)

[UEMLoad Utility for UNIX](#)

[Universal Event Monitor Server for UNIX](#)

5.4.33 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 5.27](#).

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 5.41 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](#)

5.4.34 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 UNIX is shown in [Figure 5.42](#).

```
# 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 5.42 Definition File Sample - UNIX

Components

UEMLoad Utility for UNIX

CHAPTER 6
Security

6.1 Overview

This chapter provides information on the Security feature of Indesca.

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

6.2 Security of Indesca Components

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

As the level of security rises, so does the administrative complexity of the system. Indesca 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 Command Manager](#)
- [Universal Command Server](#)
- [Universal Event Monitor Manager](#)
- [Universal Event Monitor Server](#)
- [Universal Control Manager](#)
- [Universal Control Server](#)
- [Universal Event Log Dump](#)
- [Universal Spool List](#)
- [Universal Spool Remove](#)

6.2.1 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, directories, and configuration options.
2. Account with which Universal Broker executes.
3. 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, **UNVCMDREF**; 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. User profiles that use Stonebranch products require ***CHANGE** authority to the **UNVTMP420** library.

HP NonStop

All files that the Broker creates or updates are located in either **\$SYSTEM.UNVLOG** or **\$SYSTEM.UNVTRACE**. The Broker does not need write access to its installation subvolume.

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 Solution 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 Indesca 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 [6.5 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.

Universal Broker User Account

Each Indesca component that the Universal Broker spawns inherits the Broker's account credentials. Occasionally, Indesca components must perform privileged operations, such as establishing a process execution 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 executed with a lesser-privileged user, provided that 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 Indesca server components (for example, UCMD 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 Indesca Server application file (for example, **ucmsrv** or **uemsrv**), and the Indesca Server application file must have its "set user ID on execution" bit (**setuid on exec**) set (for example, **chmod u+s ucmsrv**).

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.)

HP NonStop

Universal Broker itself does not require **super . super** privileges. For example, Universal Command Server may require **super . super** authority. Since the component inherits its user ID from the Broker, either the Broker must be running as **super . super** or the UCMD Server program must be owned by **super . super** and ProdID must be set for the server program file.

If the Broker is started as a daemon at system startup time, it is started with a user ID of **super.super**. The Broker and all its components will then have sufficient authority.

6.2.2 Universal Command Manager

File Permissions

Only trusted user accounts should have write permission to the Universal Command Manager installation directory, its subdirectories, and all files within those directories. This most likely means only an administration group should have write access.

z/OS

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

Windows

Eligible users of UCMD require read access to the message catalogs (*.umc files) in the **n1s** subdirectory of the **.\Universal** installation directory. These file permissions are set automatically during the installation.

UNIX

Eligible users of Universal Command require read access to the message catalogs (*.umc files) in the **n1s** subdirectory of the Stonebranch Solutions installation directory (**/opt/universal** by default).

IBM i

Only trusted user accounts and the **UNVUBR420** user profile should have write permission to the UCMD Server product library (**UNVPRD420**) and all files within that library.

Eligible users of UCMD require read access to the message catalogs in the **UNVNLS** file.

HP NonStop

Eligible users of Universal Command require read access to the message catalogs in the **\$SYSTEM.UNVNLS** subvolume.

RACF Protection

The UCMD Manager for z/OS verifies a users access to a RACF general resource profile. The resource profile controls a user's access to execute a command 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 Universal Command Manager RACF profiles.

Configuration Files

Only trusted user accounts should have write access to the Universal Command 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 may execute the Universal Configuration Manager (see [Section 8.4 Universal Configuration Manager](#)).

6.2.3 Universal Command Server

File Permissions

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

z/OS

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

Windows

Only trusted user accounts should have write permission to the UCMD Server installation directory and subdirectories, and all of the files within them. This most likely means only the administrator group should have write access. Eligible users of Universal Command require read access to the message catalogs (* .umc files) in the n1s subdirectory of the Stonebranch Solutions installation directory.

All eligible users of Universal Command require permission to create directories in the UCMD Server working directory, if security is activated. 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's 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 the directory.

IBM i

Only administrator accounts should have write permission to the Universal Command Server product library, **UNVPRD420**; the command reference library, **UNVCMDREF**; 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, and delete authority to these objects. As a reminder, the system value **QCRTAUT** controls public access authority to created objects unless overridden by specific commands.

Other than users authorized to use Stonebranch Solutions components, the same applies to the product temporary library, **UNVTMP420**.

HP NonStop

Only trusted user accounts should have write permission to the UCMD Server installation subvolume and all files within them.

Configuration Files

Only trusted user accounts should have write access to the Universal Command 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 may execute the Universal Configuration Manager (see [Section 8.4 Universal Configuration Manager](#)).

HP NonStop

Only trusted user accounts should have write permission to the UCMD Server configuration files, and add and delete access to the subvolume in which they reside.

Universal Command Server User ID

z/OS

Universal Command Server for z/OS requires read access to its installation data sets and its HFS working directory (defined in the component definition).

UNIX

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

See the Universal Message Translator chapter in the [Stonebranch Solutions Utilities Reference Guide](#) for details.

HP NonStop

UCMD Server requires read access to its installation subvolume. If user security is activated, the UCMD Server requires super.super access to create processes that execute with another user's identity. The UCMD Server security identity is inherited from the Universal Broker. If the Universal Broker is running with a non-super.super user ID, the UCMD Server program must have the ProgID bit set and super.super as owner. (See the Universal Message Translator chapter in the [Stonebranch Solutions Utilities Reference Guide](#) for details.)

User Authentication

User authentication is the process of verifying that a user is a known and valid user. The process used by Universal Command Server requires the user to provide an operating system-specific user name / ID and a password. The Universal Command 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 for them to do a batch log on. All users can do an interactive log on. See the [LOGON_METHOD](#) option in the [Universal Command Reference Guide](#) for more details.

UNIX

Universal Command 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 and, optionally, process session modules. 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 only available on Hewlett Packard HP-UX and Tru64 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

If the user name and password are successfully validated by the operating system, the Initiator program (`UCMSINIT`) changes the current user profile to the user profile of the user ID.

Universal Command Server User Profile

If user security is activated, the Universal Command Server for IBM i requires, by default, `*ALLOBJ` authority to change user profiles. Unless modifications are made as described in Section [Removing *ALLOBJ Authority from UNVUBR420 User Profile](#), the Server user profile, which is inherited from the Broker, requires `*ALLOBJ` authority.

Universal Command Server User ID

Universal Command Server for HP NonStop requires read access to its installation subvolume. If user security is activated, the UCMD Server requires `super.super` access to create processes that execute with another user's identity. The UCMD Server security identity is inherited from the Universal Broker. If the Universal Broker is running with a non-`super.super` user ID, the UCMD Server program must have the `ProgID` bit set and `super.super` as owner. (See Chapter [21 Universal Message Translator](#) in the [Stonebranch Solutions Utilities Reference Guide](#) for details.)

6.2.4 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.

z/OS

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 may execute the Universal Configuration Manager (see [Section 8.4 Universal Configuration Manager](#)).

6.2.5 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 may 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 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.

6.2.6 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**

HP NonStop

Eligible users of Universal Control require read access to the message catalogs in the `$SYSTEM.UNVNLS` subvolume.

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 may execute 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.

6.2.7 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.

HP NonStop

Object permissions: Only trusted user accounts should have management, existence, alter, add, update or delete authority to the Universal Control Server installation libraries and objects.

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 may 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 the [LOGON_METHOD](#) configuration option in the [Stonebranch Solutions Utilities Reference Guide](#) for more details.)

6.2.8 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.

Universal Configuration Manager

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 may execute the Universal Configuration Manager (see Section [8.4 Universal Configuration Manager](#)).

6.2.9 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](#).

6.2.10 Universal Spool Remove

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

6.3 Encryption

The [Universal Encrypt](#) utility lets you encrypt passwords for remote servers accessed by Indesca.

6.3.1 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 Indesca 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.

6.3.2 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 can also 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 Indesca, such as [Universal Access Control List](#), [X.509 Certificates](#), and . These can be further used to ensure that access to servers is properly controlled.

6.4 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 Indesca 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](#)

HP NonStop

[Creating Encrypted Files for HP NonStop](#)

6.4.1 Creating Encrypted Files for z/OS

In this example, a Universal Command 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](#)

6.4.2 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 6.1 z/OS encrypted file example

Components

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

[Universal Encrypt](#)

6.4.3 Creating Encrypted Files for Windows

In this example, a Universal Command 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

1F7DAF62583C813EA874CA168FF626C348F7BF171477D380D9A2FFFED33C539B71B4206EA502
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](#)

6.4.4 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 6.2 Windows encrypted file example

Components

[Universal Command Manager for Windows](#)

[Universal Encrypt](#)

6.4.5 Creating Encrypted Files for UNIX

In this example, a Universal Command 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

1F7DAF62583C813EA874CA168FF626C348F7BF171477D380D9A2FFFED33C539B71B4206EA502
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](#)

6.4.6 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 6.3 UNIX encrypted file example

Components

[Universal Command Manager for UNIX](#)

[Universal Encrypt](#)

6.4.7 Creating Encrypted Files for IBM i

In this example, a Universal Command command file named **MYLIB/QXTSRC (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/QXTSRC) 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](#)

6.4.8 Using Encrypted Password File on IBM i

For IBM i, the Universal Command Manager [COMMAND_FILE_ENCRYPTED](#) option ([ECMFILE](#) and [ECMMBR](#)) specifies the location of the Uencrypted file.

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

Figure 6.4 IBM i encrypted file example

Components

[Universal Command Manager for IBM i](#)

[Universal Encrypt](#)

6.4.9 Creating Encrypted Files for HP NonStop

In this example, a command file named `cmdclear` contains the following data:

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

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

```
run uencrypt /IN cmdclear, OUT cmdenc/ -key MYKEY123
```

The resulting encrypted command file is written to file `cmdenc`.

The figure below illustrates the contents of `cmdenc`.

```
# Universal Encrypt
# Created on Mon Jul 14 16:47:50 2003
# Created by uencrypt 2.1.1 Level 0

4F4813F7767318C3B1FB016F95B5FD07A6F90A787D9643A03C36503E761DF84AB64FF8877C76F9
8FDBEA1CE672A2DE943CE81BC1C159ABB01D0EC9E52E04A8C21A0269BE85F8443C1A5543901851
C29BE8223471A6BCD498163CD40D1E1866B4
```

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 IBM i](#)

[Universal Encrypt](#)

6.5 Universal Access Control List

Many Indesca 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 Indesca 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 Command 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 the [Universal Command 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](#) of the [Stonebranch Solutions Utilities Reference Guide](#) for complete details).

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

6.5.1 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 Section [Configuration File Syntax](#) for details.

HP NonStop

All UACL rules are defined in one file, **uac1cfg**. 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 located within the same subvolume as all other product configuration files.

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

6.5.2 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:

```
ucmd_request "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 6.1, 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"> • C Country name • CN Common name • L Locality • O Organization • OU Organizational Unit • ST State <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"> • <code>subject="C=US,ST=Georgia,O=Acme,CN=Road Runner"</code> • <code>subject="C=US,ST=Georgia,O=Acme,CN=Road *"</code> • <code>subject="C=US,ST=Georgia,O=Acme,CN=Road ?unner"</code> <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 emailAddress attribute of the subject field and rfc822Name of the subjectAltName extension value. Both fields format the email address as an RFC 822 addr-spec in the form of identifier@domain.</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"> • email=user1@acme.com • email=*@acme.com • email=user?@acme.com <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"> 1. dnsName of the subjectAltName extension value. 2. commonName (CN) RDN attribute of the subject field's DN value. <p>Some example HOSTNAME values are:</p> <ul style="list-style-type: none"> • hostname=bigfish.acme.com • hostname=*.acme.com <p>The values are not case sensitive.</p>
IP ADDRESS	<p>Matches the X.509 iPAddress field of the subjectAltName extension value.</p> <p>An example IPADDRESS value is:</p> <ul style="list-style-type: none"> • ipaddress=10.20.30.40
SERIAL NUMBER	<p>Matches the X.509 serialNumber value.</p> <p>The value can be specified in a hexadecimal format by prefixing the value with 0x or 0X, otherwise, the value is considered a decimal format. For example, the value 0x016A392E7F would be considered a hexadecimal format.</p> <p>An example SERIALNUMBER value is:</p> <ul style="list-style-type: none"> • serialnumber=0x7a2d52cbae

Table 6.1 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 6.2, 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"> • An asterisk (*) matches 0 or more characters and a question mark (?) matches one character. For example, AB*M matches ABCDM and ABM. AB?M matches ABCM, but not ABCDM. • The comparison is case insensitive. • 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 matches A*B. A//B matches A/B.

Table 6.2 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 6.3, 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"> • Dotted numeric form of an IP address. For example, 10.20.30.40. • Dotted numeric prefix of the IP addresses. For example, 10.20.30. matches all IP addresses starting with 10.20.30. The last dot (.) is required. • A <i>net/mask</i> expression. For example, 131.155.72.0/255.255.254.0 matches IP address range 131.155.72.0 through 131.155.73.255. The <i>mask</i> and the host value are AND'ed together. The result must match <i>net</i>. <p>Note: Contact your network administrator for calculation of the correct <i>net / mask</i> expression.</p> <ul style="list-style-type: none"> • Host name for an IP address. For example, sysa.abc.com. • Host name suffix for a range of IP addresses. For example, .abc.com matches all host names ending with abc.com, such as, sysa.abc.com. The first dot (.) is required. • A value of ALL matches all IP addresses. The value must be uppercase.
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"> • An asterisk (*) matches 0 or more characters and a question mark (?) matches one character. For example, AB*M matches ABCDM and ABM. AB?M matches ABCM but not ABCDM. • Control code <i>/c</i> switches off case-sensitivity and <i>/C</i> switches on case-sensitivity matching. The default is on. For example, /cABC matches abc. /ca/Cbc matches Abc but not ABC. • 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 matches A*B. A//B matches A/B.

Table 6.3 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 6.4, 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"> • An asterisk (*) matches 0 or more characters and a question mark (?) matches one character. For example, AB*M matches ABCDM and ABM. AB?M matches ABCM but not ABCDM. • Control code /c switches off case-sensitivity and /C switches on case-sensitivity matching. The default is on. For example, /cABC matches abc. /ca/Cbc matches Abc but not ABC. • 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 matches A*B. A//B matches A/B. • 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. <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 TOM, a LOCAL_USER value of \$RMTUSER will match if the local user name requested is also TOM. A LOCAL_USER value of \$(RMTUSER)01 will match if the local user name requested is TOM01.</p> <div data-bbox="565 1234 1133 1276" style="background-color: #f4a460; padding: 2px;">Windows</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"> • An asterisk (*) matches 0 or more characters and a question mark (?) matches one character. For example, AB*M matches ABCDM and ABM. AB?M matches ABCM but not ABCDM. • The comparison is case insensitive. • 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 matches A*B. A//B matches A/B.

Criteria	Description
REQUEST_NAME	<p>The REQUEST_NAME field matches the name of a Universal Command Manager is request. The REQUEST_NAME value has the following syntax:</p> <ul style="list-style-type: none"> • An asterisk (*) matches 0 or more characters and a question mark (?) matches one character. For example, AB*M matches ABCDM and ABM. AB?M matches ABCM but not ABCDM. • 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. • Control code /c switches off case-sensitivity and /C switches on case-sensitivity matching. The default is on. For example, /cABC matches abc. /ca/Cbc matches Abc but not ABC. • Control code /s normalizes spaces and /S 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, /sa b c matches a b c. /Sa b c matches a b c but not a bc. • 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 matches A*B. A//B matches A/B.

Table 6.4 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.

6.6 Universal Access Control List Examples

This section provides the following UACL examples.

z/OS

[Universal Broker for z/OS](#)

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

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

Windows

[Universal Broker for Windows](#)

[Universal Command Server for Windows](#)

[Universal Control Server for Windows](#)

UNIX

[Universal Broker for UNIX](#)

[Universal Command Server for UNIX](#)

[Universal Control Server for UNIX](#)

IBM i

[Universal Broker for IBM i](#)

[Universal Command Server for IBM i](#)

[Universal Control Server for IBM i](#)

HP NonStop

[Universal Broker for HP NonStop](#)

[Universal Command Server for HP NonStop](#)

[Universal Control Server for HP NonStop](#)

6.6.1 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

6.6.2 Universal Command 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.

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

ucmd_cert_access operations,*,allow,auth
ucmd_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 **SUPERID**. 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.

```
ucmd_access      10.20.30.40,TS1004,tsup1004,allow,noauth
ucmd_access      10.20.30.40,TS1004,*,allow,auth
ucmd_access      10.20.30.40,*,*,deny,auth
ucmd_access      ALL,*,superid,deny,auth

ucmd_cert_access joe,tsup1004,allow,noauth
ucmd_cert_access joe,*,allow,auth
ucmd_cert_access operations,*,deny,auth
ucmd_cert_access *,superid,deny,auth
```

Components

Universal Command Server for z/OS

6.6.3 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

6.6.4 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 6.6, below, illustrates an example.

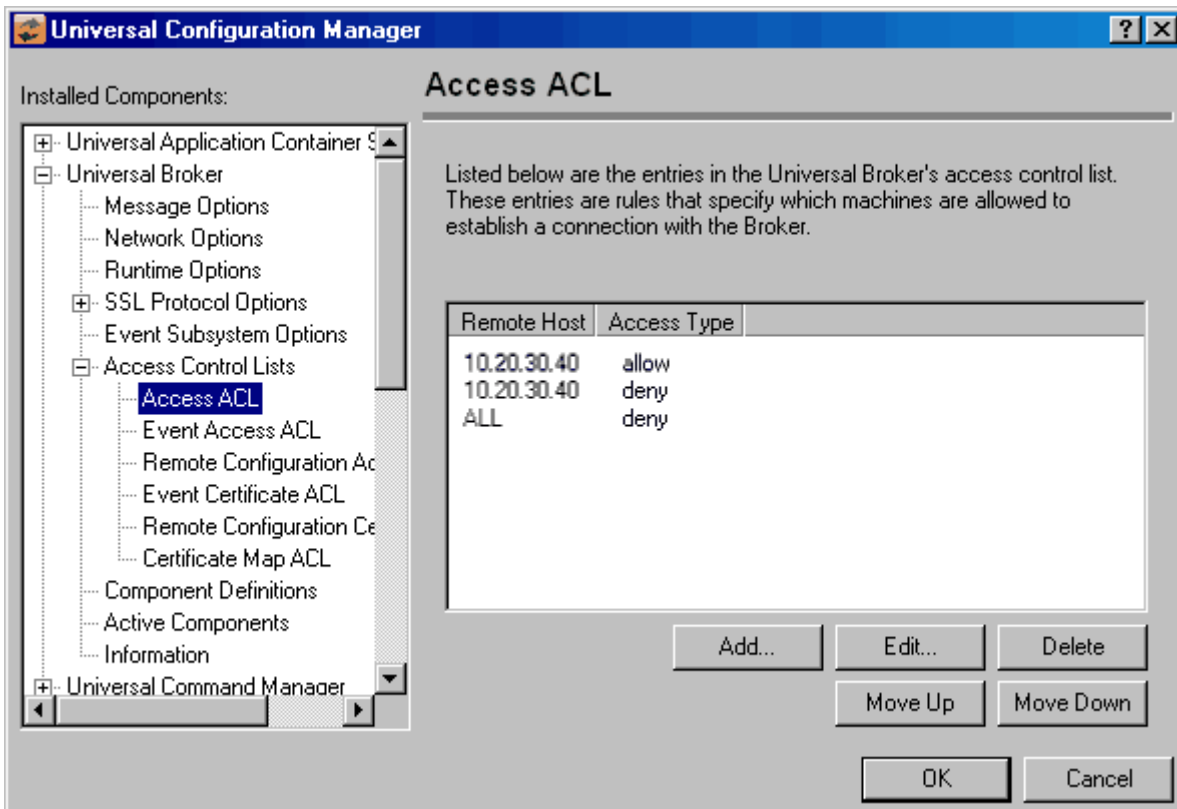


Figure 6.5 Universal Configuration Manager - Universal Broker - Access ACL

Components

Universal Broker for Windows

6.6.5 Universal Command 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 (see Section 8.4 [Universal Configuration Manager](#)). From there, ACL entries can be added, changed, deleted or sorted (rules are applied in the order in which they are listed).

Figure 6.6, 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 Command using the Administrator account on the local system, regardless of where the request originated.

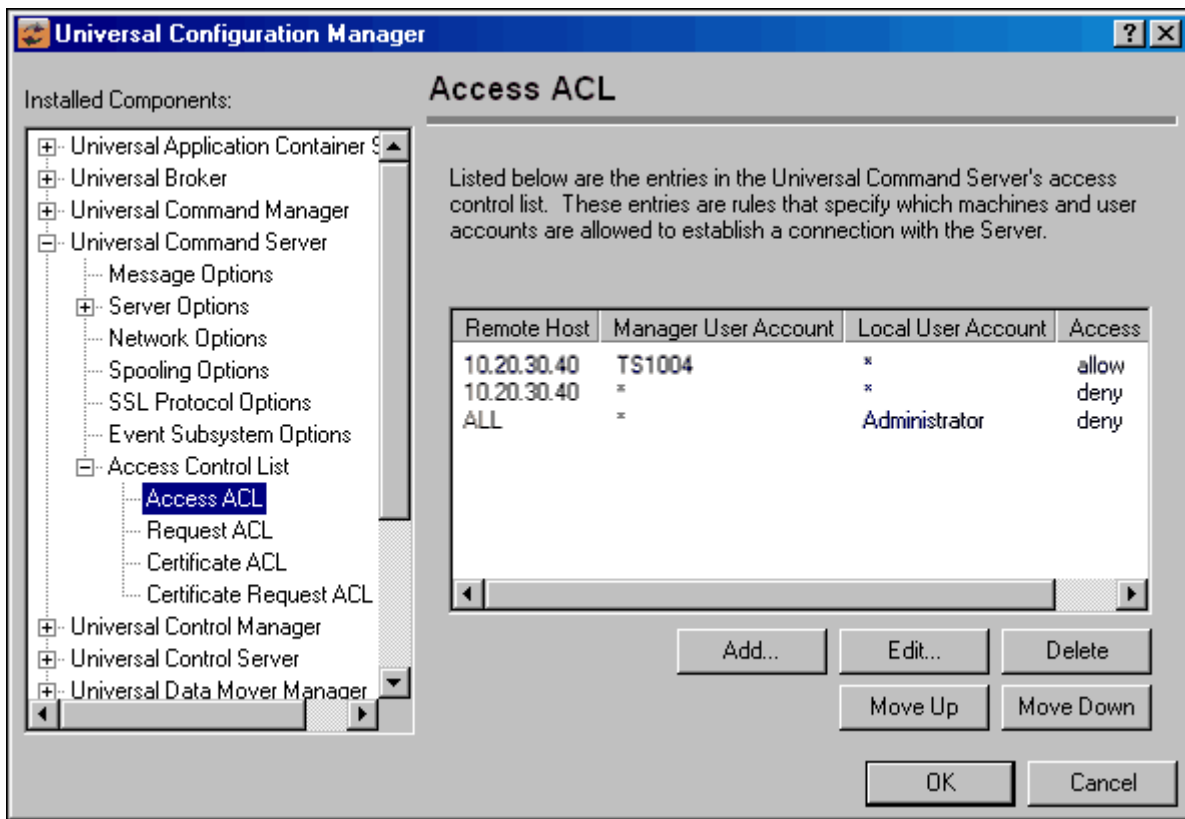


Figure 6.6 Universal Configuration Manager - Universal Command Server - Access ACL

Components

[Universal Command Server for Windows](#)

6.6.6 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 6.7, 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.

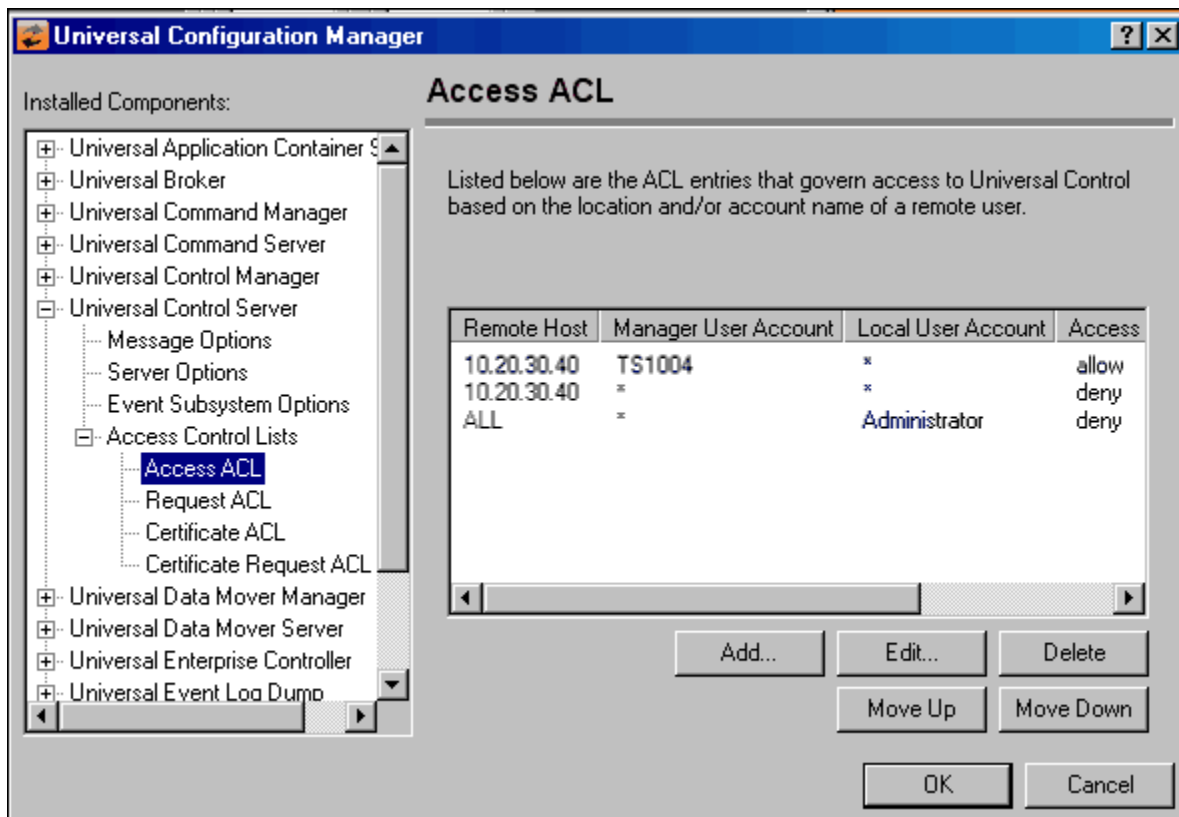


Figure 6.7 Universal Configuration Manager - Universal Control Server - Access ACL

Components

Universal Control

6.6.7 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

6.6.8 Universal Command 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.

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

ucmd_cert_access operations,*,allow,auth
ucmd_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.

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

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

Components

Universal Command Server for UNIX

6.6.9 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

6.6.10 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

6.6.11 Universal Command 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.

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

ucmd_cert_access operations,*,allow,auth
ucmd_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.

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

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

Components

Universal Command Server for IBM i

6.6.12 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

6.6.13 Universal Broker for HP NonStop

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
```

Components

Universal Broker for HP NonStop

6.6.14 Universal Command Server for HP NonStop

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

```
ucmd_access    10.20.30.,*,*,allow,auth
ucmd_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.

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

Components

Universal Command Server for HP NonStop

6.6.15 Universal Control Server for HP NonStop

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

```
uct1_access    10.20.30.,*,*,allow,auth
uct1_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.

```
uct1_access    10.20.30.40,TS1004,tsup1004,allow,noauth
uct1_access    10.20.30.40,TS1004,*,allow,auth
uct1_access    10.20.30.40,*,*,deny,auth
uct1_access    ALL,*,root,deny,auth
```

Components

Universal Control

6.7 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.

6.7.1 Sample Certificate Directory

Figure 6.8, below, provides a sample diagram of a small X.500 directory.

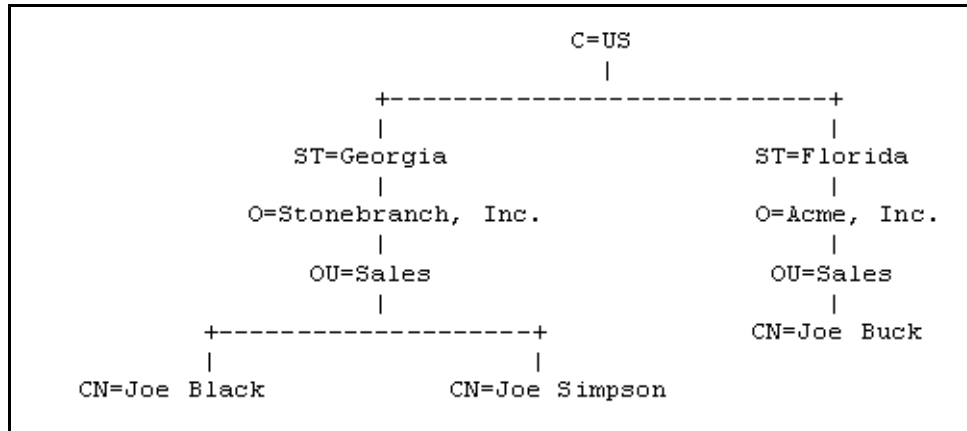


Figure 6.8 X.500 Directory (sample)

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.

6.7.2 Sample X.509 Certificate

Figure 6.9, 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 6.9 X.509 Version 3 Certificate (sample)

Note: The contents of a certificate file does not look like the information in [Figure 6.9](#), 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.

6.7.3 Certificate Fields

A certificate is composed of many fields.

[Table 6.5](#), 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 6.5 Certificate Fields

6.7.4 SSL Peer Authentication

The SSL protocol utilizes X.509 certificates to perform peer authentication. For example, a Universal Command 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.

6.8 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](#)

6.8.1 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 it to file `req.pem`
- Private key, which it writes it 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

6.8.2 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

6.9 Command References

Command references are predefined commands or scripts saved in files on a Universal Command Server system.

Universal Command Managers can request command references to be executed by specifying a `COMMAND_TYPE` value of `cmdref`. The Manager specifies the command reference by name. The Server finds the command reference file and executes the command or script contained within the command reference.

A command reference provides the ability to precisely define and control what is executed by the Server. The Manager does not provide any commands or scripts; everything is defined within the command reference. The command reference may optionally be defined to accept command or script options from the Manager.

6.9.1 Command Reference Syntax

A command reference file is a sequential file containing two sections:

1. [Options Section](#)
2. [Command Section](#)

Options Section

The Options section contains [Command Reference Options](#) that define the command reference and specify operational characteristics. Comments also can be included in the Options section.

The Options section syntax rules are:

1. Lines starting with a hash character (#) are comments.
2. Blank lines are ignored.
3. Options are specified with a Name followed by a Value.
4. Names are prefixed with a hyphen (-).
5. Names are case-insensitive.
6. Values are case-insensitive.
7. One or more spaces must separate the Name and Value.
8. Options section is ended with `<eof>` at the start of a line.

Command Reference Options

Table 6.6, below, identifies all valid command reference options.

Name	Description	Values
-format	Format of the command section. This option is required.	Valid values are: <ul style="list-style-type: none"> • cmd Single command is specified. • script Script is specified. The script is defined in the same format as if it was being defined normally in a self-contained file.
-type	Type of command or script.	Possible values depend on the Universal Command Server operating system. For example, on Windows, possible values for scripts are: <ul style="list-style-type: none"> • shell for commands • file extension with a program association for scripts. See the Universal Command Server COMMAND_TYPE and SCRIPT_TYPE options for description of valid values.
-allow_options	Specification for whether or not options provided by the Manager are passed to the command or script.	Valid values are: <ul style="list-style-type: none"> • yes Options are passed. <ul style="list-style-type: none"> • For commands, the Manager options are concatenated to the end of the command. • For scripts, the Manager options are passed to the script as command line arguments. • no Options are not passed. Default is no .

Table 6.6 Command Reference Options

Command Section

The Command section provides the actual command or script to be executed. Whether it is interpreted as a command or script is specified by the **-format** option.

Commands

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

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

Scripts

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

Command Reference Example

Figure 6.10, below, is an example command reference that executes a Windows DIR command.

Note: The two sections are separated with a logical end-of-file marker, <eof>.

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

-format          cmd
-type            shell
-allow_options   no

<eof>

DIR
```

Figure 6.10 Command Reference Syntax

6.9.2 Configuration

Command reference files are located in a location defined by the Universal Command Server `CMD_REFERENCE_DIRECTORY` option. When a Manager requests a command reference execution, the Server searches for the command reference in the configured location and executes it.

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

See the Security section of the appropriate Server platform for details.

6.9.3 Command Reference Request

A Universal Command Manager requests the execution of a command reference just as it does for other command types: via the **COMMAND_TYPE** option.

Note: Universal Command Managers earlier than version 3.1.0 must use the **SERVER_OPTIONS** option to specify a command type.

Examples

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

3.1.0 and later: no options

```
ucmd -cmd cmd100 -cmd_type cmdref ...
```

3.1.0 and later: with options

```
ucmd -cmd "cmd100 opt1 opt2" -cmd_type cmdref ...
```

Earlier than 3.1.0: no options

```
ucmd -cmd cmd100 -server " -cmd_type cmdref" ...
```

Earlier than 3.1.0: with options

```
ucmd -cmd "cmd100 opt1 opt2" -server " -cmd_type cmdref" ...
```

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

Note: For Universal Command Managers earlier than 3.1.0, the **-server** value must include at least one space between the first quotation mark and the following option (in these examples, **-cmd_type**).

Copying Files to / from Remote Systems

7.1 Overview

This chapter provides examples for the copying of files to and from remote systems.

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

z/OS

[Copy from Local z/OS to Remote Windows via Universal Copy](#)

[Copy from Remote Windows to Local z/OS via Universal Copy](#)

[Copy from Local z/OS to Remote UNIX via Universal Copy](#)

[Copy from Remote UNIX to Local z/OS via Universal Copy](#)

[Copy from Local z/OS to Remote IBM i via Universal Copy](#)

[Copy from Remote IBM i to Local z/OS via Universal Copy](#)

[Copy from Local z/OS to Remote HP NonStop via Universal Copy](#)

[Copy from Remote HP NonStop to Local z/OS via Universal Copy](#)

[Third-Party Copy via Local z/OS, from Windows to UNIX via Universal Copy](#)

[Third-Party Copy via Local z/OS, from UNIX to Windows via Universal Copy](#)

[Third-Party Copy via Local z/OS, from Windows to Windows via Universal Copy](#)

[Third-Party Copy via Local z/OS, from UNIX to UNIX via Universal Copy](#)

[Copy from Local z/OS to Remote System \(in Binary\) via Universal Copy](#)

[Copy from Remote System to Local z/OS \(in Binary\) via Universal Copy](#)

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

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

[Copy Local File from Local z/OS to Remote Windows \(with Windows Date Variables\) via Universal Copy](#)

[Copy Local File from Local z/OS to Remote UNIX \(with UNIX Data Variables\) via Universal Copy](#)

[Copy File from Remote UNIX to Local z/OS Using UNIX cat Command via Universal Command Manager for z/OS](#)

[Command Coded as a Script, Script Stored on z/OS via Universal Command Manager for z/OS](#)

Windows

[Copy from Remote UNIX to Local Windows via Universal Command Manager for Windows](#)

[Copy From Local Windows to Remote UNIX via Universal Command Manager for Windows](#)

[Copy a File from Remote UNIX to Local Windows Using the UNIX cat Command via Universal Command Manager for Windows](#)

UNIX

[Copy from Remote Windows to Local UNIX via Universal Command Manager for UNIX](#)

[Copy from Local UNIX to Remote Windows via Universal Command Manager for UNIX](#)

[Copying a File from Remote Windows to Local UNIX via Universal Command Manager for UNIX](#)

[Copying a File from Local UNIX to Remote Windows via Universal Command Manager for UNIX](#)

IBM i

[Copy from Remote Windows to Local IBM i via Universal Command Manager for IBM i](#)

[Copy from Local IBM i to Remote Windows via Universal Command Manager for IBM i](#)

[Copy a File from Remote Windows to Local IBM i via Universal Command Manager for IBM i](#)

[Copy a File from Local IBM i to Remote Windows via Universal Command Manager for IBM i](#)

[Copy a File from Remote IBM i to Local Windows via Universal Command Manager for IBM i](#)

[Copy a File from Local Windows to Remote IBM i](#)

[Copy File from Remote Windows to Local IBM i via Universal Command Manager](#)

Note: These examples reference the IBM i commands by their untagged names. If you are using commands with tagged names to run Universal Copy, substitute the tagged names for the untagged names.

HP NonStop

[Copy File from Remote Windows to Local HP NonStop via Universal Copy](#)

[Copy Local File from Local HP NonStop to Remote Windows via Universal Copy](#)

[Copy File from Remote Windows to Local HP NonStop via UCMD Manager for HP NonStop](#)

[Copy File from Remote Windows to Local HP NonStop via UCMD Manager for HP NonStop](#)

[Copy a File from Local HP NonStop to Remote Windows via UCMD Manager for HP NonStop](#)

[Copy a File from Local HP NonStop to Remote Windows via UCMD Manager for HP NonStop](#)

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

Figure 7.1, below, illustrates the copying of a file from a local z/OS system to a remote Windows system.

```
//S1 EXEC UCMDPRC
//UNVIN DD DISP=SHR,DSN=h1q.input.file
//LOGONDD DD DISP=SHR,DSN=h1q.userid(userid)
//SCRIPTDD DD *
@echo off
ucopy -mode text -output C:\OUTPUT.FILE
//SYSIN DD *
-script SCRIPTDD
-encryptedfile LOGONDD
-host dallas
/*
```

Figure 7.1 Copy from Local z/OS to Remote Windows via Universal Copy

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

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

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

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

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

[Universal Copy](#)

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

Figure 7.2, below, illustrates the copying of a file from a remote Windows system to a local z/OS system.

```
//S1 EXEC UCMDPRC
//UNVOUT DD DISP=SHR,DSN=h1q.output.file
//LOGONDD DD DISP=SHR,DSN=h1q.userid(userid)
//SCRIPTDD DD *
@echo off
ucopy -mode text C:\INPUT.FILE
//SYSIN DD *
-script SCRIPTDD
-encryptedfile LOGONDD
-host da11as
/*
```

Figure 7.2 Copy from Remote Windows to Local z/OS via Universal Copy

The JCL procedure UCMDPRC is used to execute the command. The command is sent to a remote system named **da11as** for execution. The **UNVOUT** DD specifies a local data set to use for the standard output of the remote command. The **-mode** option (value = text) is used with the **ucopy** command to force end-of-line character interpretation.

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

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

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	DD from which to read a script file. The script file is sent to the remote system for execution.
-host	Directs the command to a computer with a host name of da11as .
-encryptedfile	DD from which to read an encrypted command options file.

Components

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

[Universal Copy](#)

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

Figure 7.3, below, illustrates the copying of a file from a local z/OS system to a remote UNIX system.

```
//S1 EXEC UCMDPRC
//UNVIN DD DISP=SHR,DSN=h1q.input.file
//LOGONDD DD DISP=SHR,DSN=h1q.userid(userid)
//SCRIPTDD DD *
/opt/universal/bin/ucopy -mode text \
-output /usr/output.file
//SYSIN DD *
-script SCRIPTDD
-encryptedfile LOGONDD
-host dallas
```

Figure 7.3 Copy from Local z/OS to Remote UNIX via Universal Copy

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

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

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

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

[Universal Copy](#)

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

Figure 7.4, below, illustrates the copying of a file from a remote UNIX system to a local z/OS system.

```
//S1 EXEC UCMDPRC
//UNVOUT DD DISP=SHR,DSN=h1q.output.file
//LOGONDD DD DISP=SHR,DSN=h1q.userid(userid)
//SCRIPTDD DD *
/opt/universal/bin/ucopy -mode text \
  /usr/input.file
//SYSIN DD *
-script SCRIPTDD
-encryptedfile LOGONDD
-host dallas
/*
```

Figure 7.4 Copy from Remote UNIX to Local z/OS via Universal Copy

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

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

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

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

[Universal Copy](#)

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

Figure 7.5, below, illustrates the copying of a file from a local z/OS system to a remote IBM i system.

```
//S1 EXEC UCMDPRC
//UNVIN DD DISP=SHR,DSN=h1q.input.file
//LOGONDD DD DISP=SHR,DSN=h1q.userid(userid)
//SCRIPTDD DD *
STRUCP TOFILE(LIBRARY/OUTPUTFILE)TOMBR(MEMBER)
  CPYMODE(*TEXT)
//SYSIN DD *
-script SCRIPTDD
-encryptedfile LOGONDD
-host da11as
/*
```

Figure 7.5 Copy from Local z/OS to Remote IBM i via Universal Copy

The JCL procedure **UCMDPRC** is used to execute the command. The command is sent to a remote system named **da11as** for execution. The **TOFILE** parameter is used with the **STRUCP** command to direct the standard out to a local data set on the remote server. The **CPYMODE** option is used to force end-of-line character interpretation.

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

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

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	DD from which to read a script file. The script file is sent to the remote system for execution.
-host	Directs the command to a computer with a host name of da11as .
-encryptedfile	DD from which to read an encrypted command options file.

Components

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

[Universal Copy](#)

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

Figure 7.6, below, illustrates the copying of a file from a remote IBM i system to a local z/OS system.

```
//S1 EXEC UCMDPRC
//UNVOUT DD DISP=SHR,DSN=h1q.output.file
//LOGONDD DD DISP=SHR,DSN=h1q.userid(userid)
//SCRIPTDD DD *
STRUCP FRMFILE(LIBRARY/INPUTFILE)FRMMBR(MEMBER)
  CPYMODE(*TEXT)
//SYSIN DD *
-script SCRIPTDD
-encryptedfile LOGONDD
-host dallas
/*
```

Figure 7.6 Copy from Remote IBM i to Local z/OS via Universal Copy

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

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

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

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

[Universal Copy](#)

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

Figure 7.7, below, illustrates the copying of a file from a local z/OS system to a remote HP NonStop system.

```
//S1 EXEC UCMDPRC
//UNVIN DD DISP=SHR,DSN=h1q.input.file
//LOGONDD DD DISP=SHR,DSN=h1q.userid(userid)
//SCRIPTDD DD *
ucopy -output outputfile -mode text
//SYSIN DD *
-script SCRIPTDD
-encryptedfile LOGONDD
-host dallas
-server " -script_type OSS"
/*
```

Figure 7.7 Universal Copy for z/OS - Copy from Local z/OS to Remote HP NonStop via Universal Copy

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

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

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

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	DD from which to read a script file. The script file is sent to the remote system for execution.
-host	Directs the command to a computer with a host name of dallas .
-encryptedfile	DD from which to read an encrypted command options file.
-server " -script_type OSS"	Indicates that this is an OSS process.

Components

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

[Universal Copy](#)

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

Figure 7.8, below, illustrates the copying of a file from a remote HP NonStop system to a local z/OS system.

```
//S1 EXEC UCMDPRC
//UNVOUT DD DISP=SHR,DSN=h1q.output.file
//LOGONDD DD DISP=SHR,DSN=h1q.userid(userid)
//SCRIPTDD DD *
ucopy -mode text inputfile
//SYSIN DD *
-script SCRIPTDD
-encryptedfile LOGONDD
-host dallas
-server " -script_type OSS"
/*
```

Figure 7.8 Copy from Remote HP NonStop to Local z/OS via Universal Copy

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

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

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

SYSIN Options

The SYSIN options used in this example are:

Option	Description
-script	DD from which to read a script file. The script file is sent to the remote system for execution.
-host	Directs the command to a computer with a host name of dallas .
-encryptedfile	DD from which to read an encrypted command options file.
-server " -script_type OSS"	Indicates that this is an OSS process.

Components

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

[Universal Copy](#)

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

Figure 7.9, below, illustrates the third-party copying of a file from a local z/OS system, which executes a `ucopy` from Windows to UNIX.

```
//S1 EXEC UCMDPRC
//UNVIN DD DSN=h1q.userid(#useridunx),DISP=SHR
//LOGONDD DD DSN=h1q.userid(#useridnt),DISP=SHR
//SCRIPT DD *
@ECHO ON
:: TRANSFER FROM NT to UNIX
@SET UCOPYPATH=/opt/universal/bin/
@SET OUTPUTFILE=outputfile
@SET INPUTFILE=inputfile
@SET UNIXHOST=unixhost
@SET TEMPUNIXID=c:\temp\tempunixid
@SET MODE=text
ucopy -output %TEMPUNIXID%
ucmd-
  -cmd " %UCOPYPATH%ucopy -output %OUTPUTFILE%"-
  < %INPUTFILE% -host %UNIXHOST% -encryptedfile %TEMPUNIXID%-
  -level info -stdin -mode %MODE%
SET RC=%ERRORLEVEL%
del %TEMPUNIXID%
URC %RC%
//SYSIN DD *
-script SCRIPT
-encryptedfile LOGONDD
-host NTHOST
-level info
/*
```

Figure 7.9 Third-Party Copy via Local z/OS, from Windows to UNIX via Universal Copy

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

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

Parameters

The following parameters should be changed to match your information:

Parameter	Description
#USERIDUNIX	Encrypted userid and password member for UNIX server
#USERIDNT	Encrypted userid and password member for NT server
UCOPYPATH	Path to UCOPY on the receiving UNIX server
OUTPUTFILE	Path and filename of receiving file on UNIX server
INPUTFILE	Path and file name of sending file on Windows server
UNIXHOST	IP address or hostname of receiving UNIX server
NTHOST	IP address or hostname of sending Windows server
TEMPUNIXID	Temporary file on the Windows server used to house the encrypted logon information for the UNIX server. This file is deleted at the bottom of the script.
MODE	Mode of file transfer (binary/text).

SYSIN Options

The SYSIN options used in this example are:

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

Components

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

[Universal Command Manager for Windows](#)

[Universal Command Server for UNIX](#)

[Universal Copy](#)

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

Figure 7.10, below, illustrates the third-party copying of a file from a local z/OS system, which executes a `ucopy` from UNIX to Windows.

```
//S1 EXEC UCMDPRC
//UNVIN DD DSN=h1q.userid(#useridnt),DISP=SHR
//LOGONDD DD DSN=h1q.userid(#useridunx),DISP=SHR
//SCRIPT DD *
export UCMDPATH=/opt/universal/bin
export UCYPATH=/opt/universal/bin
export OUTPUTFILE="c:\temp\outputfile"
export INPUTFILE=/tmp/inputfile
export NTHOST=nthostname
export TEMPNTID=/tmp/tempntid
export MODE=text
$UCYPATH/ucopy -output $TEMPNTID
$UCMDPATH/ucmd \
  -cmd "ucopy -output $OUTPUTFILE"< $INPUTFILE \
  -host $NTHOST -encryptedfile $TEMPNTID -level info -stdin -mode $MODE
rc=$?
rm $TEMPNTID
exit $rc
//SYSIN DD *
-script SCRIPT
-encryptedfile LOGONDD
-host unixhostname
-level info
/*
```

Figure 7.10 Third-Party Copy via Local z/OS, from UNIX to Windows via Universal Copy

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

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

Parameters

The following parameters should be changed to match your information:

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

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

[Universal Command Manager for UNIX](#)

[Universal Command Server for Windows](#)

[Universal Copy](#)

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

Figure 7.11, below, illustrates the third-party copying of a file from a local z/OS system, which executes a `ucopy` command from Windows to Windows.

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

```
//S1 EXEC UCMDPRC
//UNVIN DD DSN=h1q.userid(#nt2logon),DISP=SHR
//LOGONDD DD DSN=h1q.userid(#ntlogon),DISP=SHR
//UNVERR DD DSN=h1q.output(stderr),DISP=SHR
//SCRIPT DD *
@ECHO ON
:: TRANSFER FROM NT to NT
@SET OUTPUTFILE=c:\temp\output.file
@SET INPUTFILE=c:\temp\input.file
@SET NT2HOST=hostname
@SET TEMPNT2ID=c:\temp\userid.enc
@SET MODE=text
ucopy -output %TEMPNT2ID%
ucmd-
  -cmd "ucopy -output %OUTPUTFILE%" <%INPUTFILE% -
  -host %NT2HOST% -encryptedfile %TEMPNT2ID% -level info -stdin -mode %MODE%
SET RC=%ERRORLEVEL%
del %TEMPNT2ID%
URC %RC%
//SYSIN DD *
  -script SCRIPT
  -encryptedfile LOGONDD
  -host NTHOST
  -level info /*
//*
//*****
//S1 EXEC PGM=UMET,PARM='-TABLE TABLE -LEVEL VERBOSE'
//STEPLIB DD DISP=SHR,DSN=h1q.UNV.SUNVLOAD
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
//TABLE DD *
  "The system cannot find the file specified." 8
```

```
/*
//SYSIN DD DISP=SHR,DSN=h1q.output(stderr)
//*****
//S1 EXEC PGM=IEBGENER
//SYSUT1 DD DISP=SHR,DSN=h1q.output(stderr)
//SYSUT2 DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSIN DD DUMMY
```

Figure 7.11 Third-Party Copy via Local z/OS, from Windows to Windows via Universal Copy

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

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

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

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

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

Parameters

The following parameters should be changed to match your information:

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

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

[Universal Command Manager for Windows](#)

[Universal Command Server for Windows](#)

[Universal Copy](#)

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

Figure 7.12, below, illustrates the third-party copying of a file from a local z/OS system, which executes a `ucopy` command from UNIX to UNIX.

```
//S1 EXEC UCMDPRC
//UNVIN DD DSN=h1q.userid(useridunxr),DISP=SHR
//LOGONDD DD DSN=h1q.userid(useridunxs),DISP=SHR
//SCRIPT DD *
export UCOPYPATH=/opt/universal/bin
export UCMDPATH=/opt/universal/bin
export OUTPUTFILE=/outputfile
export INPUTFILE=/inputfile
export UNIXRHOST=receivinghostname
export TEMPUNIXRID=/tmp/unixid.tmp
export MODE=text
$UCOPYPATH/ucopy -output $TEMPUNIXRID
$UCMDPATH/ucmd \
  -cmd "$UCOPYPATH/ucopy -output $OUTPUTFILE" < $INPUTFILE \
  -host $UNIXRHOST -encryptedfile $TEMPUNIXRID -level info -stdin -mode $MODE
rc=$?
rm $TEMPUNIXRID
exit $rc
//SYSIN DD *
  -script SCRIPT
  -encryptedfile LOGONDD
  -host unixshost
  -level info
/*
```

Figure 7.12 Third-Party Copy via Local z/OS, from UNIX to UNIX via Universal Copy

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

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

Parameters

The following parameters should be changed to match your information:

Parameter	Description
UCOPYPATH	Path pointing to the ucopy executable on the second UNIX server
UCMDPATH	Path pointing to the ucmd executable on the second UNIX server
OUTPUTFILE	Path and filename of receiving file
INPUTFILE	Path and file name of sending file
UNIXSHOST	IP address or hostname of sending UNIX server
UNIXRHOST	IP address or hostname of receiving UNIX server
TEMPUNIXRID	Temporary file on the sending UNIX server used to house the encrypted logon information for the receiving UNIX server.
MODE	Mode of file transfer (binary / text). Default is set to text.
USERUNXR	Points to the userid / password information for the receiving UNIX server.
USERUNXS	Points to the userid / password information for the sending UNIX server.

SYSIN Options

The SYSIN options used in this example are:

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

Components

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

[Universal Command Manager for UNIX](#)

[Universal Command Server for UNIX](#)

[Universal Copy](#)

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

Figure 7.13, below, illustrates the copying of a file from a local z/OS system to a remote system, in binary, with no end-of-line character interpretation.

```
//S1 EXEC UCMDPRC
//UNVIN DD DISP=SHR,DSN=h1q.input.file
//LOGONDD DD DISP=SHR,DSN=h1q.userid(userid)
//SCRIPTDD DD *
@echo off
ucopy -output C:\OUTPUT.FILE
//SYSIN DD *
-script SCRIPTDD
-encryptedfile LOGONDD
-host dallas
-stdin -mode binary
/*
```

Figure 7.13 Copy from Local z/OS to Remote System (in Binary) via Universal Copy

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

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

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

[Universal Copy](#)

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

Figure 7.14, below, illustrates the copying of a file from a remote system to a local z/OS system, in binary, with no end-of-line character interpretation.

```
//S1 EXEC UCMDPRC
//UNVOUT DD DISP=SHR,DSN=h1g.output.file
//LOGONDD DD DISP=SHR,DSN=h1q.userid(userid)
//SCRIPTDD DD *
@echo off
ucopy C:\INPUT.FILE
//SYSIN DD *
-script SCRIPTDD
-encryptedfile LOGONDD
-host dallas
-stdout -mode binary
/*
```

Figure 7.14 Copy from Remote System to Local z/OS (in Binary) via Universal Copy

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

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

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

[Universal Copy](#)

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

Figure 7.15, below, illustrates the copying of a file from a local z/OS system to a remote z/OS system (with encryption, compression, and data authentication).

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

Figure 7.15 Copy from Local z/OS to Remote z/OS (with Encryption, Compression, and Data Authentication) via Universal Copy

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

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

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

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

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

[Universal Copy](#)

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

Figure 7.16, below, illustrates the copying of a file from a remote z/OS system to a local z/OS system (with encryption, compression, and data authentication).

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

Figure 7.16 Copy from Remote z/OS to Local z/OS (with Encryption, Compression, and Data Authentication) via Universal Copy

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

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

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

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

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

[Universal Copy](#)

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

Figure 7.17, below, illustrates the copying of a file from a local z/OS system to a remote Windows system.

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

```
//S1 EXEC UCMDPRC
//UNVIN DD DISP=SHR,DSN=h1q.input.file
//LOGONDD DD DISP=SHR,DSN=h1q.userid(userid)
//SCRIPTDD DD *
@echo off
for /f "tokens=1 delims=/" %a in ('date /t') do set daymm=%a
for /f "tokens=2" %a in ('echo %daymm%') do set mm=%a
for /f "tokens=2 delims=/" %a in ('date /t') do set dd=%a
for /f "tokens=3 delims=/" %a in ('date /t') do set yy=%a
echo daymm: %daymm%
echo mmddy: %mm%%dd%%yy%
ucopy -output c:\temp\outputfile%mm%%dd%%yy%
//SYSIN DD *
-script SCRIPTDD
-encryptedfile LOGONDD
-host dallas
/*
```

Figure 7.17 Copy from Local z/OS File to Remote Windows (with Windows Date Variables) via Universal Copy

The JCL procedure **UCMDPRC** is used to execute the command. The command is sent to a remote system named **dallas** for execution. The **-output** option is used with the **ucopy** command to direct stdout to a local data set on the remote Windows server. The file name is created with a date variable. The date variable is set to the current date in the commands preceding the **ucopy** command.

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

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

SYSIN Options

The SYSIN options used in this example are:

Option	Description
<code>-script</code>	DD from which to read a script file. The script file is sent to the remote system for execution.
<code>-host</code>	Directs the command to a computer with a host name of da11as .
<code>-encryptedfile</code>	DD from which to read an encrypted command options file.

Components

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

[Universal Command Server for Windows](#)

[Universal Copy](#)

7.1.18 Copy Local File from Local z/OS to Remote UNIX (with UNIX Data Variables) via Universal Copy

Figure 7.18, below, illustrates the copying of a file from a local z/OS system to a remote UNIX system. The file name on the UNIX server is dynamically created based on the current date.

```
//S1 EXEC UCMDPRC
//UNVIN DD DISP=SHR,DSN=h1q.input.file
//LOGONDD DD DISP=SHR,DSN=h1q.userid(userid)
//SCRIPTDD DD *
DATEN=`date +%d%m`
export DATEN
echo $DATEN
/opt/universal/bin/ucopy \
  -output /tmp/output$DATEN.file
//SYSIN DD *
-script SCRIPTDD
-encryptedfile LOGONDD
-host dallas
```

Figure 7.18 Copy from Local z/OS File to Remote UNIX (with UNIX Date Variables) via Universal Copy

The JCL procedure **UCMDPRC** is used to execute the command. The command is sent to a remote system named **dallas** for execution. The stdout redirection character **>** is used with the **ucopy** command to direct stdout to a local data set on the remote server. The file name is created with a date variable, which is set to the current date in the commands preceding the **ucopy** command. The path to the **ucopy** binary must be specified if the directory is not defined in the user's path environmental variable.

Additional command line options are read from the encrypted file allocated to DD **LOGONDD**. The UNIX continuation character **** is used to split the **ucopy** command to two lines.

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

SYSIN Options

The SYSIN options used in this example are:

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

Components

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

[Universal Command Server for UNIX](#)

[Universal Copy](#)

7.1.19 Copy File from Remote UNIX to Local z/OS Using UNIX cat Command via Universal Command Manager for z/OS

```

//UNIXCAT JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*
//          JCLLIB ORDER=username.UNV.SUNVSAMP
//*
//STEP1    EXEC UCMDPRC,
//          STDOUT='DISP=SHR,DSN=username.UNIX.FILE1'
//SYSIN    DD *
-cmd 'cat /export/home/username/file1'
-host Unix_1 -userid username -pwd password -level audit
/*

```

Figure 7.19 UNIX Copy to Local z/OS Dataset via Universal Command Manager for z/OS

SYSIN Options

The SYSIN options used in this example are:

Option	Description
- cmd	Remote command cat /export/home/username/file1 to execute. The cat program copies the files specified on the command line to its stdout.n .
- host	Directs the command to a computer with host address Unix_1 .
- userid	Remote user ID with which to execute the command.
- pwd	Password for the user ID.
- level	Message level output for this command execution.

Components

Universal Command Manager for z/OS

7.1.20 Command Coded as a Script, Script Stored on z/OS via Universal Command Manager for z/OS

Figure 7.20, below, illustrates how to code the remote command as a script.

The benefits are:

- Multiple commands can be executed with one UCMD Manager.
- Script can be housed separate from the execution JCL.
- Script can be stored and promoted via change control procedures.
- Provides separation of the remote command from UCMD SYSIN.

```
//UNISCRI JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//          JCLLIB ORDER=SBI.UNV.SUNVSAMP
//*
//STEP1    EXEC UCMDPRC
//UNVIN    DD DUMMY
//UNVOUT   DD SYSOUT=*
//UNVERR   DD SYSOUT=*
//SCRIPTDD DD DISP=SHR,DSN=username.JCL.CNTRL(USCRIPT)
//SYSIN    DD *
-host          Unix_1
-userid       username
-pwd          password
-script       SCRIPTDD
/*
//
```

Figure 7.20 Command Coded as a Script via Universal Command Manager for z/OS

The JCL procedure **UCMDPRC** is used to execute the command. The command is sent to remote system **Unix_1** for execution. The process will authenticate and run under the authority of user id **username**.

SYSIN Options

The SYSIN options used in this example are:

Option	Description
- host	Directs the command to a computer with host address Unix_1 .
- userid	Remote user ID with which to execute the command.
- pwd	Password for the user ID.
- script	DD from which to read a script file, which is sent to remote system for execution.

Components

Universal Command Manager for z/OS

7.1.21 Copy from Remote UNIX to Local Windows via Universal Command Manager for Windows

[Figure 7.21](#), below, illustrates the copying of a file from a remote UNIX system to a local Windows system. Although the command is shown on two lines, it should be entered on one line at the command prompt. If it is coded in a script, the Windows continuation character of `^` must be used.

```
ucmd -cmd " /opt/universal/bin/ucopy unixinputfile"
-host unixhost -encryptedfile unixid.file > c:\temp\ntoutputfile
```

Figure 7.21 Copy from Remote UNIX to Local Windows via Universal Command Manager for Windows

The standard out of the `ucopy` command on the remote host is redirected back to the local host and written to `c:\temp\ntoutputfile`. The command `ucopy` is installed as part of Universal Command Server on the remote system.

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

Parameters

The following parameters should be changed to match your information:

Parameter	Description
ntoutputfile	Path and filename of output file
unixinputfile	Path and file name of input file
unixhost	IP address of remote UNIX server
unixid.file	File on the Windows server used to house the authentication parameters for the UNIX server

Command Line Options

The command line options used in this example are:

Option	Description
<code>-cmd</code>	Remote command <code>ucopy file</code> to execute. The <code>ucopy</code> program copies the files specified on the command line to its STDOUT.
<code>-host</code>	Directs the command to a computer with a host name of <code>unixhost</code> .
<code>-encryptedfile</code>	File from which to read encrypted command options.

Components

[Universal Command Manager for Windows](#)

[Universal Copy](#)

7.1.22 Copy From Local Windows to Remote UNIX via Universal Command Manager for Windows

Figure 7.22, below, illustrates the copying of a file from a local Windows system to a remote UNIX system. Although the command is shown on two lines, it should be entered on one line at the command prompt. If it is coded in a script, the Windows continuation character of `^` must be used.

```
ucmd -cmd " /opt/universal/bin/ucopy -output /tmp/unixoutputfile"
-host unixhost -encryptedfile unixid.file
< c:\temp\ntinputfile
```

Figure 7.22 Copy from Local Windows to Remote UNIX via Universal Command Manager for Windows

The stdin of the `ucmd` manager on the local host is redirected to the stdout of the remote host and written to `/tmp/unixoutputfile`. The command `ucopy` is installed as part of Universal Command Server on the remote system. The file is copied as a text file since the default transfer mode for standard files is text.

Parameters

The following parameters should be changed to match your information:

Parameter	Description
unixoutputfile	Path and filename of output file
ntinputfile	Path and file name of input file
unixhost	IP address of sending UNIX server
unixid.file	File on the Windows server used to house the authentication parameters for the UNIX server

Command Line Options

The command line options used in this example are:

Option	Description
-cmd	Remote command <code>ucopy file</code> to execute. The <code>ucopy</code> program copies the files specified on the command line to its stdout.
-host	Directs the command to a computer with a host name of <code>unixhost</code> .
-encryptedfile	File from which to read encrypted command options.

Components

[Universal Command Manager for Windows](#)

[Universal Copy](#)

7.1.23 Copy a File from Remote UNIX to Local Windows Using the UNIX `cat` Command via Universal Command Manager for Windows

This example copies a file from a remote UNIX system to a local file.

[Figure 7.23](#), below, illustrates the command. Although the command is shown on two lines, it should be entered as one line at the command prompt.

```
ucmd -cmd "cat ~/file" -host dallas  
      -userid joe -pwd password -comment "copy ~/file from dallas" > localfile
```

Figure 7.23 Copy from Remote UNIX to Windows using UNIX `cat` Command via UCMD Manager for Windows

The stdout of the `cat` command on the remote host is redirected back to the local host and written to the stdout of `ucmd`, which is then redirected to the local file `localfile`.

Command Line Options

The command line options used in this example are:

Option	Description
<code>-cmd</code>	Remote command - " <code>cat ~/file</code> " - to execute. The <code>cat</code> program copies the files specified on the command line to its stdout.
<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 command.
<code>-pwd</code>	Password for the user ID.
<code>-comment</code>	Description of the process executed by Universal Command.

The file is copied as a text file, since the default transfer mode is `text`.

Components

[Universal Command Manager for UNIX](#)

[Universal Copy](#)

7.1.24 Copy from Remote Windows to Local UNIX via Universal Command Manager for UNIX

[Figure 7.24](#), below, illustrates the copying of a file from a remote Windows system to a local UNIX server.

If it is coded in a script, then the UNIX continuation character of `\` must be used.

```
ucmd -cmd 'ucopy ntinputfile' -host nthost -encryptedfile
ntid.file > /tmp/unixoutputfile
```

Figure 7.24 Copy from Remote Windows to Local UNIX via Universal Command Manager for UNIX

The stdout of the `ucopy` command on the remote host is redirected back to the local host and written to `/tmp/unixoutputfile`. The command `ucopy` is installed as part of Universal Command Server on the remote system.

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

Command Line Options

The command line options used are:

Option	Description
<code>-cmd</code>	Remote command <code>ucopy file</code> to execute. The <code>ucopy</code> program copies the files specified on the command line to its stdout.
<code>-host</code>	Directs the command to a computer with a host name of <code>nthost</code> .
<code>-encryptedfile</code>	File from which to read encrypted command options.

Components

[Universal Command Manager for Windows](#)

[Universal Copy](#)

7.1.25 Copy from Local UNIX to Remote Windows via Universal Command Manager for UNIX

Figure 7.25, below, illustrates the copying of a file from a local UNIX system to a remote Windows server.

Although the command is shown on two lines, it should be entered on one line at the command prompt. If it is coded in a script, the UNIX continuation character of `\` must be used.

```
ucmd -cmd 'ucopy -output c:\temp\ntoutput.file' -host nhost
-encryptedfile login.file < /tmp/unixinput.file
```

Figure 7.25 Copy from Local UNIX to Remote Windows via Universal Command Manager for UNIX

The stdin of the `ucmd` manager on the local host is redirected to the remote host and written to stdout file `c:\temp\ntoutput.file`. The command `ucopy` is installed as part of Universal Command Server on the remote system.

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

Command Line Options

The command line options used are:

Option	Description
<code>-cmd</code>	Remote command <code>ucopy file</code> to execute. The <code>ucopy</code> program copies the files specified on the command line to its stdout.
<code>-host</code>	Directs the command to a computer with a host name of <code>nhost</code> .
<code>-encryptedfile</code>	File from which to read encrypted command options.

Components

[Universal Command Manager for UNIX](#)

[Universal Copy](#)

7.1.26 Copying a File from Remote Windows to Local UNIX via Universal Command Manager for UNIX

This example copies a file from a remote Windows system to a local file.

[Figure 7.26](#), below, illustrates the command. Although the command is shown on two lines, it should be entered as one line at the command prompt.

```
ucmd -cmd 'ucopy file' -host dallas
      -userid joe -pwd password > localfile
```

Figure 7.26 Copy from Remote Windows to UNIX via Universal Command Manager for UNIX

The stdout of the `ucopy` command on the remote host is redirected back to the local host and written to the stdout of `ucmd`, which is then redirected to the local file `localfile`.

The command `ucopy` is installed as part of UCMD Server on the remote system.

The file is copied as a text file since the default transfer mode is `text`.

Command Line Options

The command line options used in this example are:

Option	Description
<code>-cmd</code>	Remote command <code>ucopy file</code> to execute. The <code>ucopy</code> program copies the files specified on the command line to its stdout.
<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 command.
<code>-pwd</code>	Password for the user ID.

Components

[Universal Command Manager for Windows](#)

[Universal Copy](#)

7.1.27 Copying a File from Local UNIX to Remote Windows via Universal Command Manager for UNIX

This example copies a local file to a remote Windows system.

Figure 7.27, below, illustrates the command. Although the command is shown on two lines, it should be entered as one line at the command prompt.

```
ucmd -cmd 'ucopy > remotefile' -host dallas
      -userid joe -pwd password < localfile
```

Figure 7.27 Copy from UNIX to Remote Windows via Universal Command Manager for UNIX

The `ucopy` command receives its stdin file from `ucmd`. The standard in of UCMD is redirected from `localfile`.

The command `ucopy` is installed as part of Universal Command Server on the remote system.

The file is copied as a text file, since the default transfer mode is `text`.

Command Line Options

The command line options used in this example are:

Option	Description
<code>-cmd</code>	Remote command <code>ucopy > remotefile</code> to execute. The <code>ucopy</code> program copies its standard in to its standard out.
<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 command.
<code>-pwd</code>	Password for the user ID.

Components

[Universal Command Manager for UNIX](#)

[Universal Copy](#)

7.1.28 Copy from Remote Windows to Local IBM i via Universal Command Manager for IBM i

Figure 7.28, below, illustrates the copying of a file from a remote Windows system to a local IBM i file.

```
STRUCM CMD('ucopy c:\ntinput.file') HOST(nthost) USERID(joe) PWD(akksdiq)
SOTFILE(library/outputfile) SOTMBR(member)
```

Figure 7.28 Copy from Remote Windows to Local IBM i via Universal Command Manager for IBM i

The stdout of the **ucopy** command on the remote host is redirected back to the local host and written to the stdout of **ucmd**, which is directed to the local file **SOTFILE** and, optionally, **SOTMBR**.

The command **ucopy** is installed as part of Universal Command Server on the remote system.

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

Command Line Options

The command line options used are:

Option	Description
CMD	Remote command ucopy file to execute. The ucopy program copies the files specified on the command line to its stdout.
HOST	Directs the command to a computer with a host name of nthost .
USERID	Remote user ID with which to execute the command.
PWD	Password for the user ID.
SOTFILE / SOTMBR	Location to which the stdout file data is written.

Components

[Universal Command Manager for Windows](#)

[Universal Copy](#)

7.1.29 Copy from Local IBM i to Remote Windows via Universal Command Manager for IBM i

Figure 7.29, below, illustrates the copying of a local IBM i file to a remote Windows system.

```
STRUCM CMD('ucopy -output c:\ntoutput.file') HOST(nthost) USERID(joe)
PWD(akksdiq) SINFILE(library/inputfile) SINMBR(member)
```

Figure 7.29 Copy from Local IBM i to Remote Windows via Universal Command Manager for IBM i

The **ucopy** command receives its stdin file from **ucmd**. The stdin of **ucmd** is redirected from **SINFILE** and, optionally, **SINMBR** to stdout of **ucopy**, which is **c:\ntoutput.file**.

The command **ucopy** is installed as part of Universal Command Server on the remote system.

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

Command Line Options

The command line options used are:

Option	Description
CMD	Remote command ucopy > file to execute. The ucopy program copies its stdin to its stdout.
HOST	Directs the command to a computer with a host name of nthost .
USERID	Remote user ID with which to execute the command.
PWD	Password for the user ID.
SINFILE / SINMBR	Location from which the stdin file data is written.

Components

[Universal Command Manager for IBM i](#)

[Universal Copy](#)

7.1.30 Copy a File from Remote Windows to Local IBM i via Universal Command Manager for IBM i

This example copies a file from a remote Windows system to a local file.

Figure 7.30, below, illustrates the command.

```
STRUCM CMD('ucopy file') HOST(dallas)
      USERID(joe) PWD(password) SOTFILE(localfile)
```

Figure 7.30 Copy from Remote Windows to IBM i via Universal Command Manager for IBM i

The standard out of the **ucopy** command on the remote host is redirected back to the local host and written to the file specified by **SOTFILE**.

The command **ucopy** is installed as part of UCMD Server on the remote system.

The file is copied as a text file, since the default transfer mode is **text**.

Command Line Options

The command line options used in this example are:

Option	Description
CMD	Remote command ucopy file to execute. The ucopy program copies the files specified on the command line to its stdout.
HOST	Directs the command to a computer with a host name of dallas .
USERID	Remote user ID with which to execute the command.
PWD	Password for the user ID.
SOTFILE	Local file that receives the stdout of the remote command.

Components

[Universal Command Manager for IBM i](#)

[Universal Copy](#)

7.1.31 Copy a File from Local IBM i to Remote Windows via Universal Command Manager for IBM i

This example copies a local file to a remote Windows system.

[Figure 7.31](#), below, illustrates the command.

```
STRUCM CMD('ucopy > remotefile') HOST(dallas)
      USERID(joe) PWD(password) SINFILE(localfile)
```

Figure 7.31 Copy from IBM i to Remote Windows via Universal Command Manager for IBM i

The `ucopy` command receives its standard in file from STRUCM. The standard in of STRUCM is read from `localfile`, as specified by `SINFILE`.

The command `ucopy` is installed as part of UCMD Server on the remote system.

The file is copied as a text file since the default transfer mode is `text`.

Command Line Options

The command line options used in this example are:

Option	Description
<code>CMD</code>	Remote command <code>ucopy > remotefile</code> to execute. The <code>ucopy</code> program copies its stdin to its stdout.
<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 command.
<code>PWD</code>	Password for the user ID.
<code>SINFILE</code>	Local file that receives the stdin of the remote command.

Components

[Universal Command Manager for IBM i](#)

[Universal Copy](#)

7.1.32 Copy a File from Remote IBM i to Local Windows via Universal Command Manager for IBM i

This example shows a file copy initiated by Windows which copies the first member of a file from IBM i to a file on the Windows system.

Figure 7.32, below, illustrates the command.

```
ucmd -host sysName -userid userId -pwd password
-cmd "strucp frmfile(mylib/myfile)" > D:\tmp\File400.txt
```

Figure 7.32 Copy from Remote IBM i to Windows via Universal Command Manager for IBM i

UCMD running on Windows invokes STRUCP via a UCMD Server running on IBM i. The FRMFILE parameter overrides input from stdin to the file `mylib/file`. Since the FRMMBR parameter is not used, input defaults to the file member `*FIRST`. Data is transferred from `mylib/myfile` to `D:\tmp\File400.txt` via UCMD Manager stdout.

The command `STRUCP` is installed as part of UCMD Server on the IBM i system.

The file is copied as a text file, since the default transfer mode is `text`.

Command Line Options

The command line options used in this example are:

Option	Description
<code>-cmd</code>	Remote command to execute on the IBM i.
<code>-host</code>	Directs the command to a computer with a host name of <code>sysName</code> .
<code>-userid</code>	IBM i user ID with which to execute the command.
<code>-pwd</code>	Password for the user ID.

Components

[Universal Command Manager for Windows](#)

[Universal Command Server for IBM i](#)

[Universal Copy](#)

7.1.33 Copy a File from Local Windows to Remote IBM i

This example shows a file copy initiated by Windows which copies a file from Windows to the first member of a file on IBM i.

Figure 7.33, below, illustrates the command.

```
ucmd -host sysName -userid userId -pwd password
-cmd "strucp tofile(mylib/readme)" < D:\tmp\README.txt
```

Figure 7.33 Copy from Windows to Remote IBM i Using Universal Copy on IBM i

Using redirected stdin, UCMD Manager, running under Windows sends, transfers data to a UCMD Server running on the remote IBM i system, **sysName**. The UCMD Server on **sysName** invokes UCOPY to transfer the data to **mylib/readme.mylib/readme** file member ***FIRST** is used since **TOMBR** was not specified.

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

The file is copied as a text file since the default transfer mode is **text**.

Command Line Options

The command line options used in this example are:

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

Components

[Universal Command Manager for Windows](#)

[Universal Command Server for IBM i](#)

[Universal Copy](#)

7.1.34 Copy File from Remote Windows to Local IBM i via Universal Command Manager

Figure 7.34, below, illustrates the copying of a file from a remote Windows system to a local file.

```
STRUCM CMD('ucopy file') HOST(dallas) USERID(joe) PWD(password)
SOTFILE(localfile)
```

Figure 7.34 Copy from Remote Windows to Local IBM i via Universal Command Manager for IBM i

UCOPY, which the UCMD Server invokes on system `da11as`, sends data to the UCMD Manager running under IBM i using standard output. The UCMD Manager, in turn, receives input via stdin and writes to file `localfile`.

Command Line Options

The command line options used in this example are:

Option	Description
<code>CMD</code>	Remote command <code>ucopy file</code> to execute.
<code>HOST</code>	Directs the command to a computer with a host name of <code>da11as</code> .
<code>USERID</code>	Remote user ID with which to execute the command.
<code>PWD</code>	Password for the user ID.
<code>SOTFILE</code>	Local file that receives the stdout of the remote command.

Components

[Universal Command Manager for IBM i](#)

[Universal Command Server for Windows](#)

[Universal Copy](#)

7.1.35 Copy File from Remote Windows to Local HP NonStop via Universal Copy

Figure 7.35, below, illustrates the copying of a file from a remote Windows system to a local file.

Although the command shown is on two lines, it should be entered on one line at the command prompt.

The HP NonStop manager is executed within the TACL environment.

```
run $SYSTEM.UNV.BIN.ucmd /OUT outputfile/ -cmd 'ucopy inputfile' -host dallas
-userid joe -pwd akksdiq
```

Figure 7.35 Copy from Remote Windows to Local File via Universal Copy for HP NonStop

The stdout of the **ucopy** command on the remote host is redirected back to the local host and written to the stdout of **ucmd**, which is then redirected to the local file **outputfile**. The command **ucopy** is installed as part of Universal Command Server on the remote system.

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

Command Line Options

The command line options used are:

Option	Description
-cmd	Remote command ucopy file to execute. The ucopy program copies the files specified on the command line to its stdout.
-host	Directs the command to a computer with a host name of dallas .
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

Components

[Universal Command Manager for HP NonStop](#)

[Universal Command Server for Windows](#)

[Universal Copy](#)

7.1.36 Copy Local File from Local HP NonStop to Remote Windows via Universal Copy

Figure 7.36, below, illustrates the copying of a local file to a remote Windows system.

Although the command is shown on two lines, it should be entered on one line at the command prompt.

The HP NonStop manager is executed within the TACL environment.

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

```
run $SYSTEM.UNVBIN.ucmd /IN inputfile/ -cmd 'ucopy -output outputfile'
-host dallas -userid joe -pwd akksdiq
```

Figure 7.36 Copy Local File to Remote Windows via Universal Copy for HP NonStop

The **ucopy** command receives its stdin file from **ucmd**. The stdin of **ucmd** is redirected from **inputfile**. The command **ucopy** is installed as part of Universal Command Server on the remote system.

Command Line Options

The command line options used are:

Option	Description
-cmd	Remote command ucopy to execute. The ucopy program copies stdin to stdout.
-host	Directs the command to a computer with a host name of dallas .
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

Components

[Universal Command Manager for HP NonStop](#)

[Universal Command Server for Windows](#)

[Universal Copy](#)

7.1.37 Copy File from Remote Windows to Local HP NonStop via UCMD Manager for HP NonStop

Figure 7.37, below, illustrates the command that copies a file from a remote Windows system to a local file.

Although the command is shown on two lines, it should be entered on one line at the command prompt.

```
run ucmd -cmd 'ucopy file' -host dallas -server " -script_type oss"
        -userid joe -pwd password -stdout -localfile localfile
```

Figure 7.37 Copy from Remote Windows to HP NonStop via Universal Command Manager for HP NonStop

The stdout of the `ucopy` command on the remote host is redirected back to the local host and written to the standard out of UCMD, which is then redirected to the local file `localfile`.

The command `ucopy` is installed as part of UCMD Server on the remote system.

The file is copied as a text file, since the default transfer mode is `text`.

Command Line Options

The command line options used in this example are:

Option	Description
<code>-cmd</code>	Remote command <code>ucopy file</code> to execute. The <code>ucopy</code> program copies the files specified on the command line to its stdout.
<code>-host</code>	Directs the command to a computer with a host name of <code>dallas</code> .
<code>-server</code>	Command line options for the UCMD Server process. The value <code>-script_type oss</code> is specified to notify the UCMD Server that it is to execute an OSS process, since Universal Copy is a native OSS program.
<code>-userid</code>	Remote user ID with which to execute the command.
<code>-pwd</code>	Password for the user ID.
<code>-stdout</code>	Start of the stdout options.
<code>-localfile</code>	Filename to which to redirect output.

Components

[Universal Command Manager for HP NonStop](#)

[Universal Command Server for Windows](#)

[Universal Copy](#)

7.1.38 Copy File from Remote Windows to Local HP NonStop via UCMD Manager for HP NonStop

Figure 7.38, below, illustrates how to copy a file from a remote Windows system to a local file.

Although the command is shown on two lines, it should be entered on one line at the command prompt.

```
run $SYSTEM.UNVBIN.ucmd -cmd 'ucopy file' -host dallas -userid joe
-pwd password -stdout -localfile localfile
```

Figure 7.38 Copy Remote Windows File to Local HP NonStop via Universal Command Manager for HP NonStop

The standard out of the `ucopy` command on the remote host is redirected back to the local host and written to the standard out of UCMD, which then is redirected to the local file `localfile`. The command `ucopy` is installed as part of UCMD Server on the remote system. The process will authenticate and run under the authority of `userid joe`.

Command Line Options

The command line options used in this example are:

Option	Description
<code>-cmd</code>	Remote command <code>ucopy file</code> to execute.
<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 command.
<code>-pwd</code>	Password for the user ID.
<code>-stdout</code>	Start of the stdout options.
<code>-localfile</code>	Filename to which to redirect output.

Components

[Universal Command Manager for HP NonStop](#)

[Universal Command Server for Windows](#)

[Universal Copy](#)

7.1.39 Copy a File from Local HP NonStop to Remote Windows via UCMD Manager for HP NonStop

This example copies a local file to a remote Windows system.

Figure 7.39, below, illustrates the command. Although the command is shown on multiple lines, it should be entered as one line at the command prompt.

```
run ucmd -cmd 'ucopy > remotefile' -host dallas
      -server " -script_type OSS" -userid joe -pwd password
      -stdin -localfile localfile
```

Figure 7.39 Copy from HP NonStop to Remote Windows via Universal Command Manager for HP NonStop

The **ucopy** command receives its standard in file from UCMD. The standard in of UCMD is redirected from **localfile**.

The command **ucopy** is installed as part of UCMD Server on the remote system.

The file is copied as a text file since the default transfer mode is **text**.

Command Line Options

The command line options used in this example are:

Option	Description
-cmd	Remote command ucopy > remotefile to execute. The ucopy program copies it stdin to its stdout.
-host	Directs the command to a computer with a host name of dallas .
-server	Command lines options for the UCMD Server process. The value -script_type OSS is specified to notify the UCMD Server that it is to execute an OSS process, since universal copy is a native OSS program.
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-stdin	Start of the stdout options.
-localfile	Filename to which to redirect output.

Components

[Universal Command Manager for HP NonStop](#)

[Universal Command Server for Windows](#)

[Universal Copy](#)

7.1.40 Copy a File from Local HP NonStop to Remote Windows via UCMD Manager for HP NonStop

Figure 7.40, below, illustrates how to copy a local file to a remote Windows system.

Although the command is shown on two lines, it should be entered on one line at the command prompt.

```
run $SYSTEM.UNV.BIN.ucmd -cmd 'ucopy > remotefile' -host dallas -userid joe
-pwd password -stdin -localfile localfile
```

Figure 7.40 Copy Local File from to Remote Windows via Universal Command Manager for HP NonStop

The `ucopy` command receives its standard in file from the UCMD `localfile` parameter. The file is written to `remotefile` on the remote system. The command `ucopy` is installed as part of UCMD Server on the remote system. The process will authenticate and run under the authority of `userid joe`.

Command Line Options

The command line options used in this example are:

Option	Description
<code>-cmd</code>	Remote command <code>ucopy file</code> to execute.
<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 command.
<code>-pwd</code>	Password for the user ID.
<code>-stdin</code>	Start of the stdin options.
<code>-localfile</code>	Filename from which to redirect input.

Components

[Universal Command Manager for HP NonStop](#)

[Universal Command Server for Windows](#)

[Universal Copy](#)

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 Stonebranch Solutions 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, Windows, and HP NonStop

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

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

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

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

Example command line options follow:

Short form:

```
-l info -G yes
```

Long form:

```
-level info -login yes  
-LEVEL info -LOGiN YES
```

IBM i

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

Example command line options follow:

Command line options:

```
MSGLEVEL(INFO) COMPRESS(*YES)
```

All 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)
```

HP NonStop

Environment variables in HP NonStop are defined with HP NonStop Advanced Command Language (TACL) commands for the current job environment.

Example of setting an environment variable:

Set option UCMDLEVEL to a value of INFO:

```
PARAM UCMDLEVEL INFO
```

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 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.

Seeion [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.

HP NonStop

The configuration files on HP NonStop are stored as EDIT files, file code 101, within the **\$SYSTEM.UNVCONF** subvolume. The files can be edited with the EDIT editor.

See [Configuration File Syntax](#) 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 components can be configured remotely by Universal Enterprise Controller using the I-Management Console client application, and can be "locked down" so that they *only* can be remotely configured.

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

Universal Broker supports remote configuration in either of two modes:

1. [Unmanaged Mode](#)
2. [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 4.2.0 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.

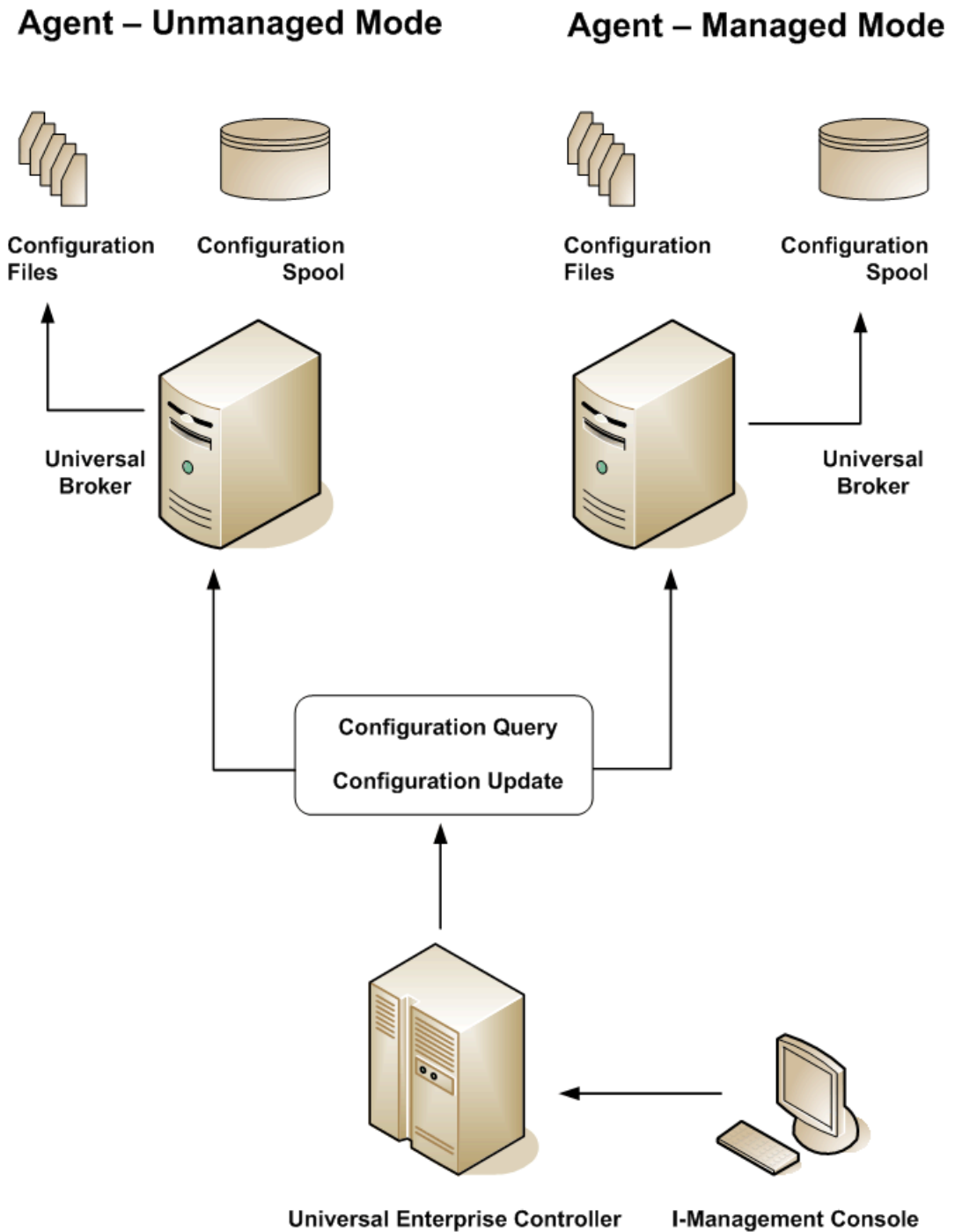


Figure 8.1 Remote Configuration - Unmanaged and Managed Modes of Operation

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:

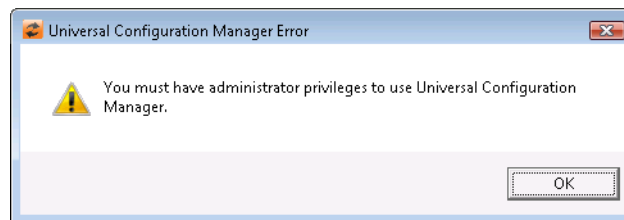


Figure 8.2 Universal Configuration Manager Error dialog – Windows Vista / Windows 7

2. Click **OK** to dismiss the error message.

The Windows Vista / Windows 7 Program Compatibility Assistant (PCA) displays the following dialog:

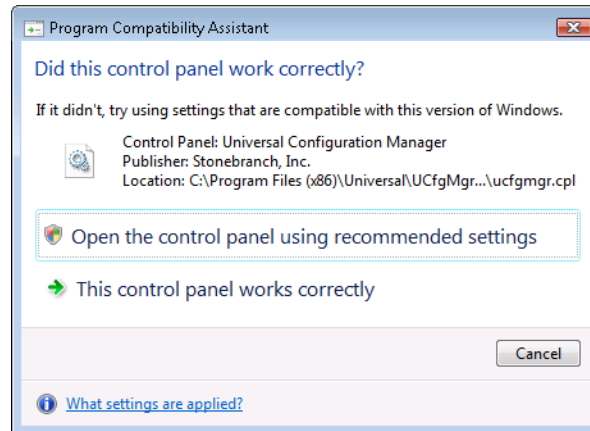


Figure 8.3 Program Compatibility Assistant – Windows Vista / Windows 7

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.

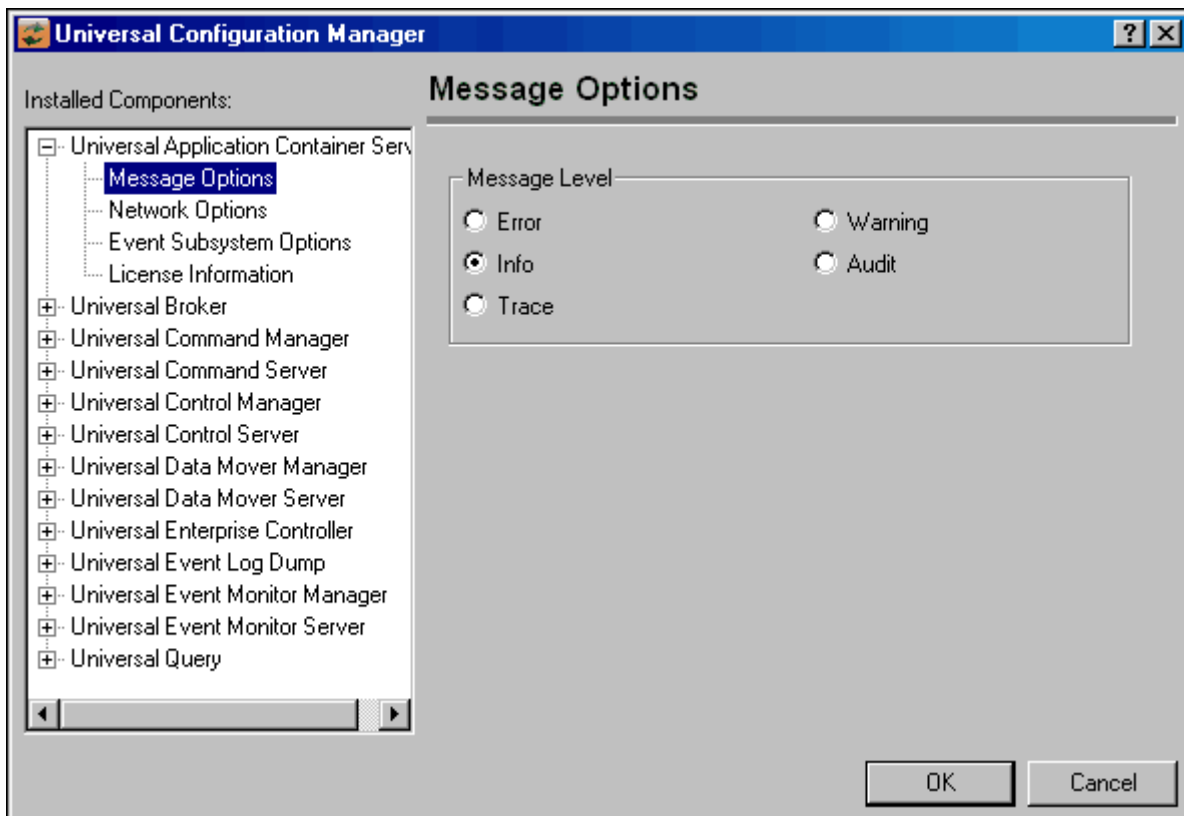


Figure 8.4 Universal Configuration Manager

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 Command Installed Components

Universal Command Manager

Figure 8.5 illustrates the Universal Configuration Manager screen for the Universal Command Manager.

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

The text describes the selected component, Universal Command Manager.

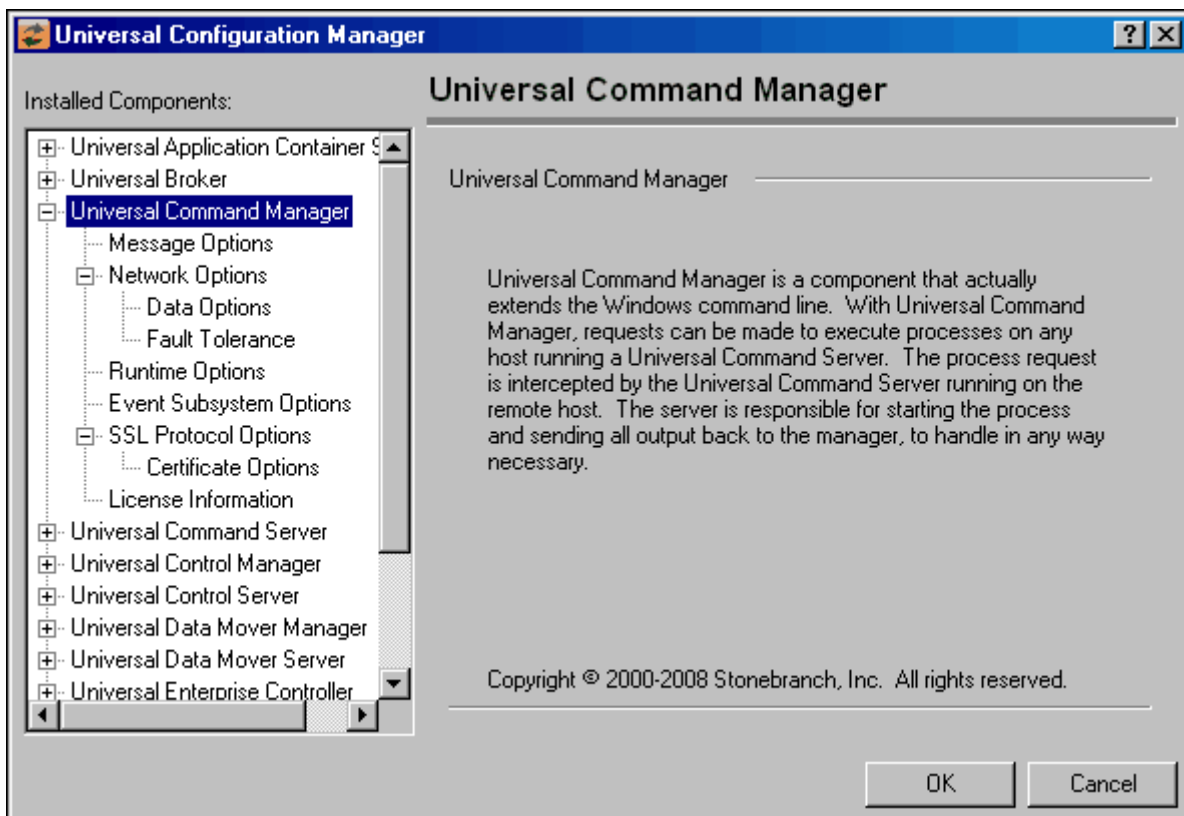


Figure 8.5 Universal Configuration Manager - UCMD Manager

Universal Command Server

Figure 8.6 illustrates the Universal Configuration Manager screen for the Universal Command Server.

The Installed Components list identifies all of the UCMD Server property pages.

The text describes the selected component, Universal Command Server.

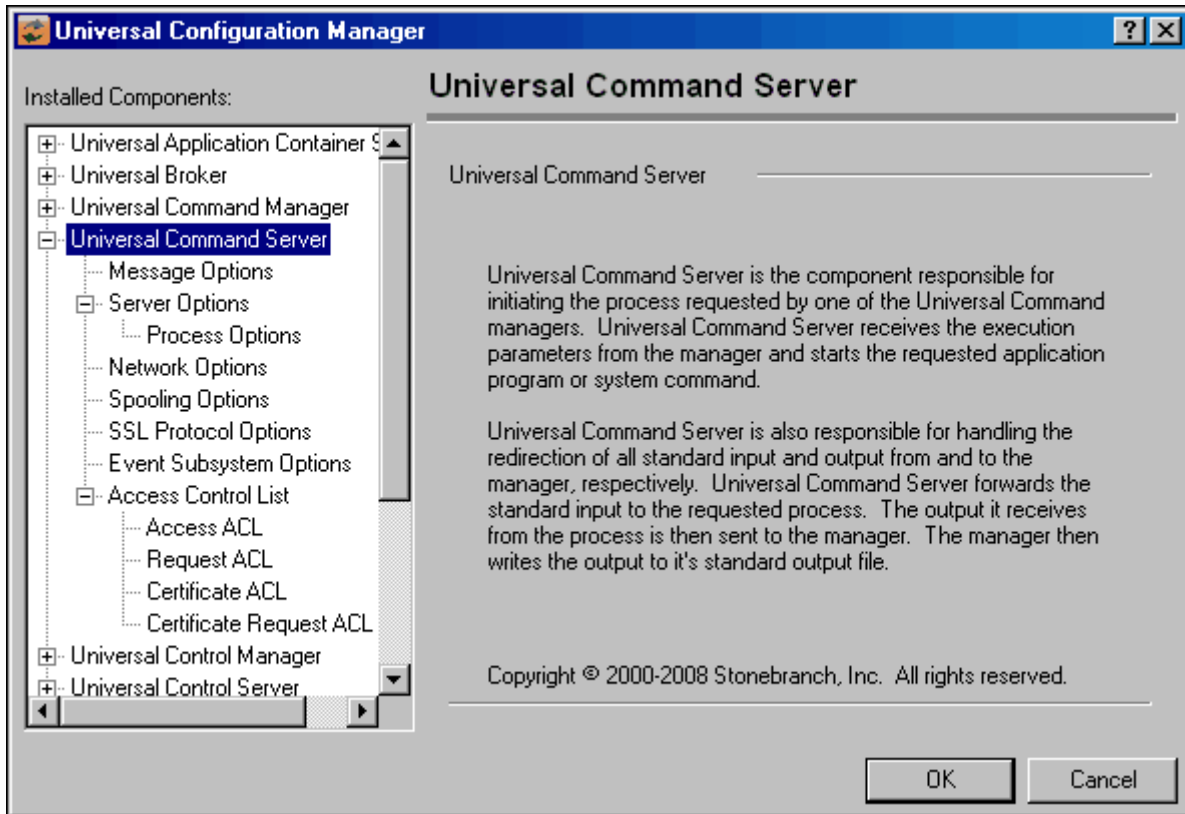


Figure 8.6 Universal Configuration Manager - UCMD Server

8.4.8 Universal Command Agent for SOA Installed Components

Universal Application Container Server

Figure 8.7 illustrates the Universal Configuration Manager screen for the Universal Application Container Server.

The Installed Components list identifies all of the UAC Server property pages.

The text describes the selected component, Universal Application Container Server.

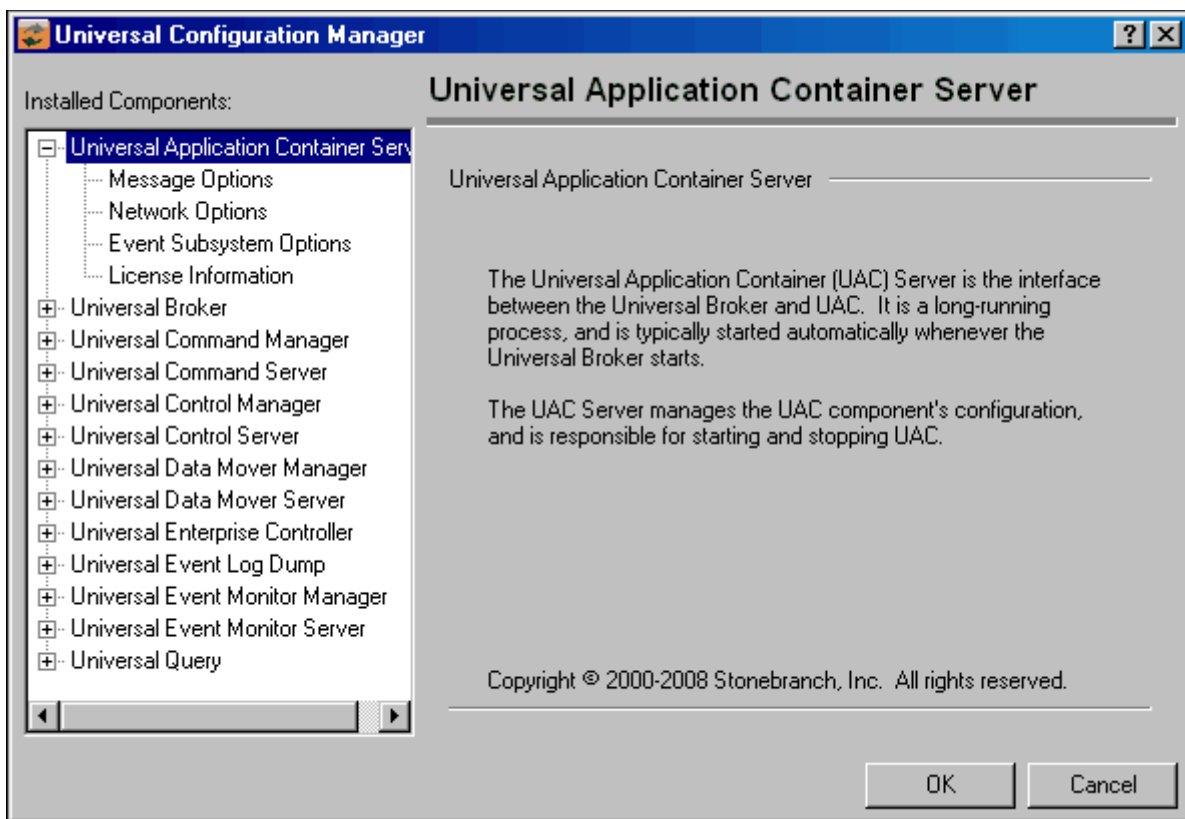


Figure 8.7 Universal Configuration Manager - UAC Server

8.4.9 Universal Event Monitor Installed Components

Universal Event Monitor Manager

Figure 8.8 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.

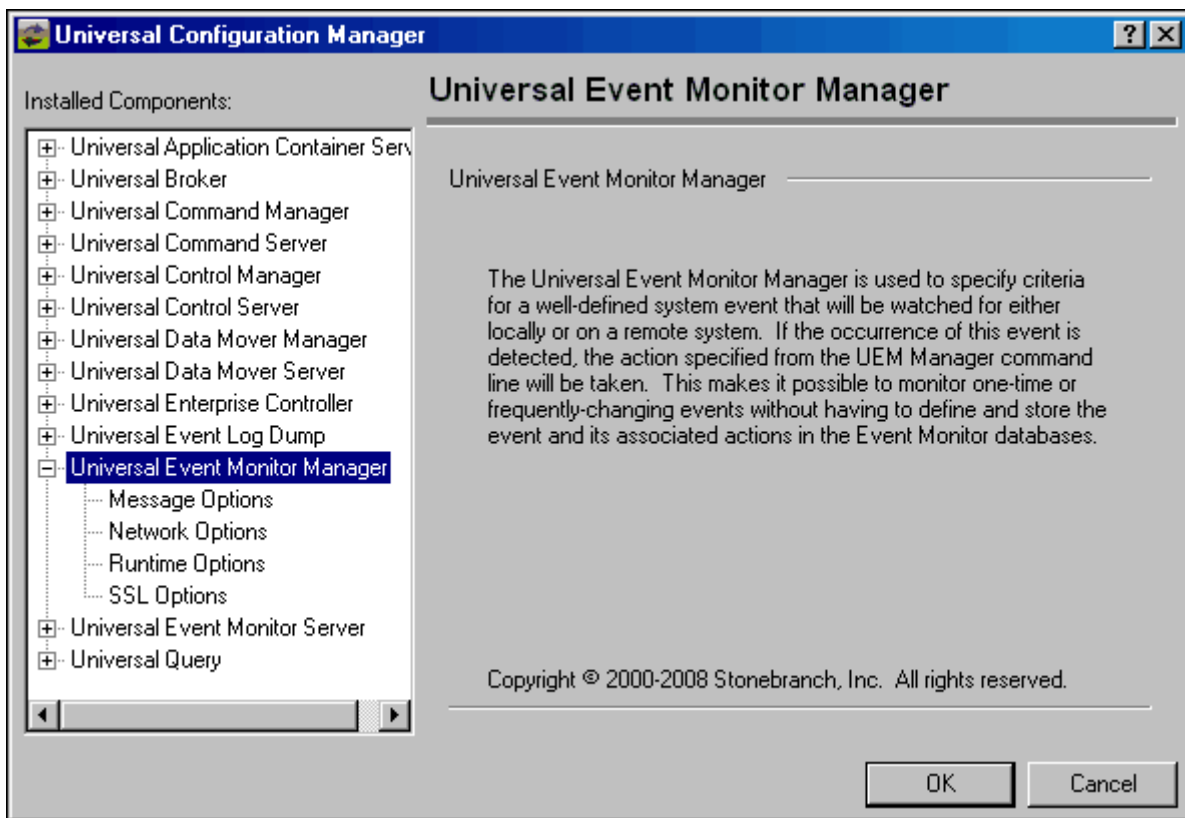


Figure 8.8 Universal Configuration Manager - UEM Manager

Universal Event Monitor Server

Figure 8.9 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.

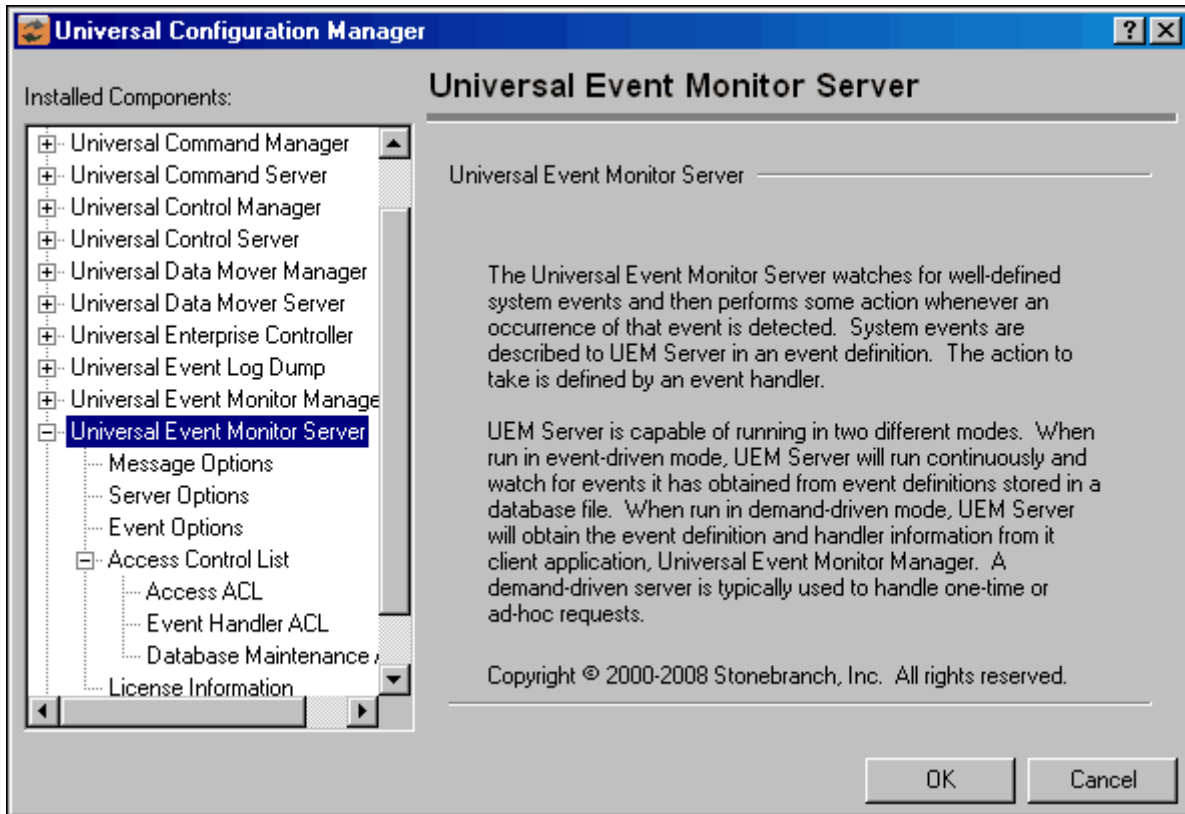


Figure 8.9 Universal Configuration Manager - UEM Server

8.4.10 Universal Enterprise Controller Component

Figure 8.10 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.

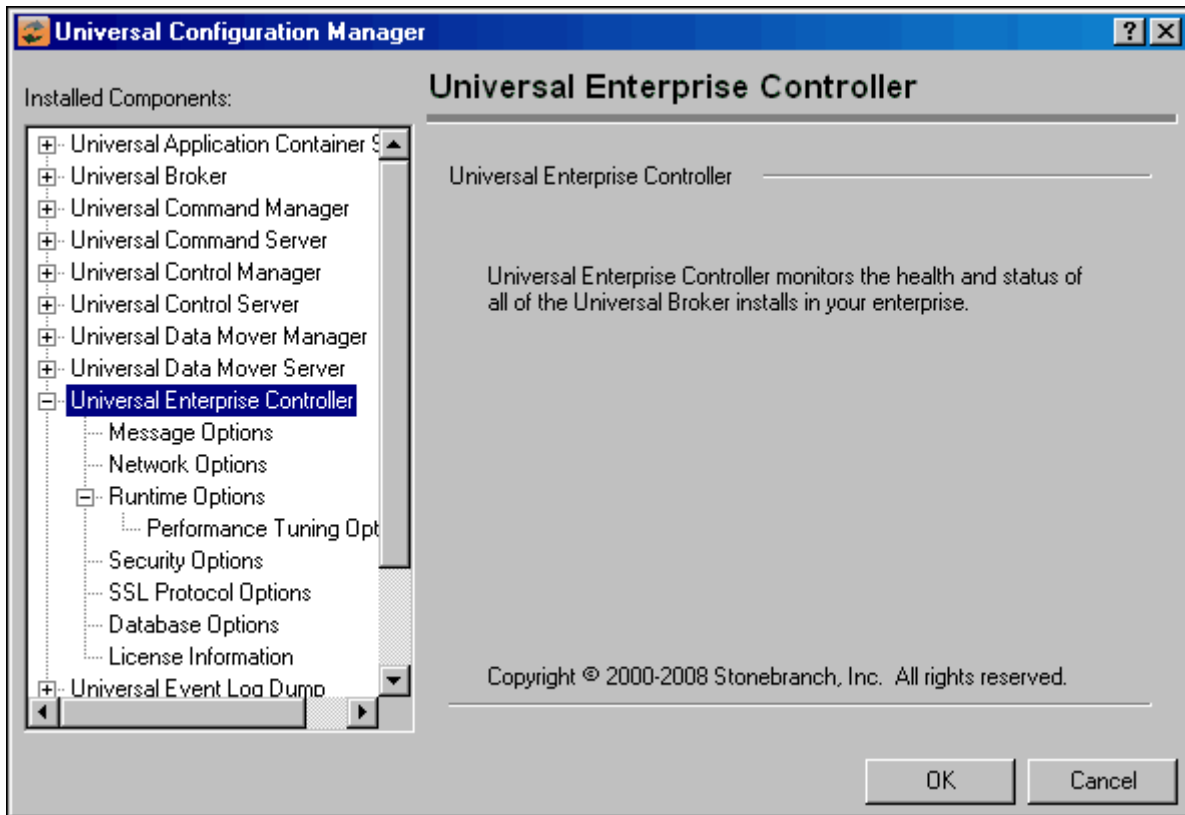


Figure 8.10 Universal Configuration Manager - Universal Enterprise Controller

8.4.11 Universal Broker Installed Component

Figure 8.11 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.

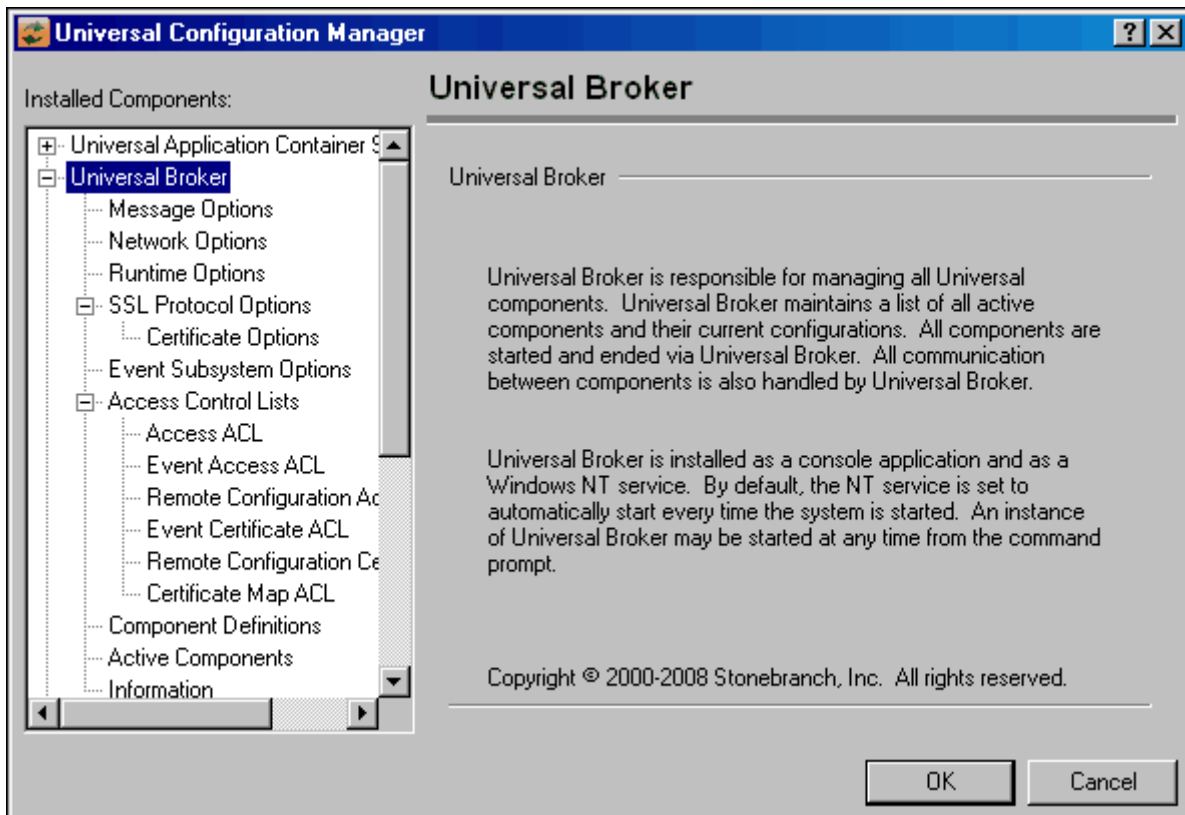


Figure 8.11 Universal Configuration Manager - Universal Broker

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 Indesca 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](#)

HP NonStop

[Refreshing Universal Broker via Universal Control from HP NonStop](#)

[Refreshing a Component via Universal Control from HP NonStop](#)

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 8.12 Refreshing Universal Broker via Universal Control from z/OS

This example refreshes the Universal Broker configuration on host **dallas**.

The refresh request 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 UNVCONF . The Broker replaces all component definitions with the newly read component definitions. New component definitions are added and deleted component definitions are removed.
Step 3	Read the Universal Access Control List configuration file allocated to ddname UNVACL . The Broker replaces its UACL entries with the newly read entries.

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 dallas .

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 8.13 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 dallas .
<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 8.14 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 dallas .
<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 8.15 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 8.16 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"> <code>MESSAGE_LANGUAGE</code> <code>RUNNING_MAX</code>
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 8.17 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 8.18 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 dallas .
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 UNVCONF and member UBROKER .
Step 2	Read all component definitions found in the component definition file, UNVPRD420 / UNVCOMP . 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 UNVCONF and member UACL . 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 8.19 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

8.6.9 Refreshing Universal Broker via Universal Control from HP NonStop

This example refreshes Universal Broker on a remote system.

```
run uctl -refresh -host dallas -userid joe -pwd akksdiq
```

Figure 8.20 Refreshing Universal Broker via Universal Control from HP NonStop

Command Line Options

The command line options used in this example are:

Option	Description
-refresh	Type of component to refresh on the remote system.
-host	Directs the command to a computer with a host name of dallas .
-userid	Remote user ID with which to execute the Universal Control Server process.
-pwd	Password for the user ID.

The REFRESH command directs Universal Broker to take the following actions:

Step	Procedure
Step 1	Read its configuration file UBRCFG . Universal Broker refreshes the following configuration options: <ul style="list-style-type: none"> • MESSAGE_LANGUAGE • RUNNING_MAX
Step 2	Read all component definitions found in the component definition subvolume. 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 UACL CFG . The Broker replaces its UACL entries with the newly read entries.

Components

Universal Control

8.6.10 Refreshing a Component via Universal Control from HP NonStop

This example refreshes a component on a remote system.

```
run uct1 -refresh uems -cmdid "UEM-dallas" -host dallas -userid joe  
-pwd akksdiq
```

Figure 8.21 Refreshing Component via Universal Control from HHP NonStop

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

8.7 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 8.22, 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 8.22 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 8.23, 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 8.23 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 8.24, 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\UCmdMgr"
```

Figure 8.24 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 Indesca component management:

- [Starting 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 Indesca Server component (for Universal Command, Universal Event Monitor, Universal Control, and Universal Application Container) has a Component Definition.

The Component Definition is a text file of options that defines component-specific information required by the Universal Broker.

Each Component Definition defines the following type of information:

- Component type (for Universal Event Monitor Servers only).
- Component name.
- Component command name.
- Component configuration file name.
- Component working directory path.
- Number of component instances that can run simultaneously.
- Specification for whether or not the component starts automatically when the Universal Broker starts.

The reference guide for each component contains detailed information about its Component Definition.

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 Section [9.3 Starting Components](#)).

For a complete explanation of the difference between demand-driven and event-driven Universal Event Monitor Servers, see Section [7.2 Demand-Driven vs. Event-Driven](#) in the [Universal Event Monitor Reference Guide](#).

9.3 Starting Components

There are four ways in which Indesca components are started.

Starting Manually

The following components are started manually and run in the background until they are stopped manually:

- Universal Broker
- Universal Enterprise Controller

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 Command Server
- Universal Control Server
- Universal Event Monitor Server (demand-driven)

Starting Automatically

The following components are auto-start components; that is, they start automatically when the Universal Broker starts and run until they are stopped manually:

- Universal Application Container Server
- Universal Event Monitor Server (event-driven)

Note: A Universal Event Monitor Server Component Definition also can specify that an event-driven server is not started automatically (see [Starting via Universal Control](#), below).

Starting via Universal Control

Universal Control can start Server components that do not require interaction with a Manager. Currently, only one Indesca component can be started via Universal Control:

- Universal Event Monitor Server (event-driven)

9.4 Stopping Components

Any Indesca 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 (UEC) client application, to stop running any Indesca 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](#)

HP NonStop

[Starting Universal Broker for HP Nonstop](#)

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 regarding the security requirements of Universal Broker and all Indesca components, see [Chapter 6 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 stop request followed by a start 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 6 Security](#) for details on their security requirements for specific Stonebranch Solutions 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 within the context of any one Stonebranch-defined subsystem.

The subsystem pre-start job entries define jobs that are in an initialized state. They are not executing but are ready to accept a request and execute at any time. Pre-starting jobs before they are required improves the overall throughput of the subsystem jobs.

Universal Broker jobs running under **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

9.5.5 Starting Universal Broker for HP Nonstop

Universal Broker for HP NonStop runs as an Open System Services (OSS) application.

It can be executed in two different environments:

- [Console Application](#)
- [Daemon](#)

Differences between the environments are described in the following sections.

Only one instance of the Universal Broker can execute at any one time. A PID file is used to help ensure only one active instance. The PID file is a locking mechanism that prevents the execution of a second Broker. The PID file, named **UBRPID**, is created in subvolume **\$SYSTEM.UNVLOG** by default. If the PID file is in the PID subvolume, it is assumed that a Universal Broker instance is executing.

Console Application

The command **ubroker** starts Universal Broker as a console application.

[Figure 9.5](#), below, illustrates the Universal Broker start command.

```
ubroker [OPTIONS...]
```

Figure 9.5 Universal Broker for HP NonStop - Start Command

Console Security

The Universal Broker runs with the same user ID as the user who starts it. The Universal Broker does not require **super . super** rights. It only requires access to its installation subvolume and files.

However, components started by Universal Broker also run with the same user ID as Universal Broker. Some components may require **super . super** rights.

(See the security documentation of the components you wish to run for details on their security requirements.)

Daemon

Universal Broker can run as a daemon process. This is the preferred method of running the Broker. A daemon startup script is provided to manage the starting and stopping of the Broker daemon. The startup script utilizes the PID file to ensure only one instance of the Broker is executing at any one time. For this reason, the startup script should be used to start and stop the Broker.

Note: The Universal Broker daemon startup script and the Universal Broker daemon program file both are installed within the `$$SYSTEM.UNVBIN` subvolume. The Broker daemon startup script name is `ubrokerd` and the Broker daemon program file name is `ubrd`.

```
ubrokerd { start | stop | status | restart }
```

Figure 9.6 Universal Broker for HP NonStop - Daemon Startup Script Syntax

[Table 9.2](#), below, describes the command line arguments to the Universal Broker daemon startup.

Command	Description
start	Starts the Universal Broker daemon. Only one instance of Universal Broker can run at any given time, so if the Broker is already running, the command fails and the script returns.
stop	Stops the Universal Broker daemon. If the Broker daemon is not running, the script simply returns.
Status	Returns the status of the Universal Broker daemon: either <i>running</i> or <i>stopped</i> . If the daemon is running, the script displays its process ID.
Restart	Performs a stop request followed by a start request.

Table 9.2 Universal Broker for HP NonStop - Command Line Arguments to Daemon Startup Script

Daemon Security

When a daemon is started at system initialization, it is started as user `super . super`. The `super . super` user ID provides sufficient authority for the Broker and any component it may start.

If the daemon is started with a non-super user ID, the environment is the same as if it was started as a console application (see [Console Security](#)).

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 Indesca 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](#)

HP NonStop

Stopping an HP NonStop 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 9.7 Universal Control for z/OS - Start Component Example

SYSIN Options

The SYSIN options used in this example are:

Option	Description
<code>-start</code>	Name of the component to start on the remote system.
<code>-cmdid</code>	Assigns a command identifier of " UEM-da11as " to the started component.
<code>-host</code>	Directs the command to a computer with a host name of da11as .
<code>-userid</code>	Remote user ID with which to execute the Universal Control Server process. The started component, in fact, will execute with the Universal Broker's security context.
<code>-pwd</code>	Password for the user ID.

Components

Universal Control

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 9.8 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
-stop	Component to stop.
-host	Directs the command to a computer with a host name of da11as .
-userid	Remote user ID with which to execute the stop request.
-pwd	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 9.9 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 9.10 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 dallas .
-userid	Remote user ID with which to execute the command. This must match the user ID originally used to start the component.
-pwd	Password for the user ID.

Components

Universal Control

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 9.11 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 9.12 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 dallas .
-userid	Remote user ID with which to execute the command. This must match the user ID originally used to start the component.
-pwd	Password for the user ID.

Components

Universal Control

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 9.13 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 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

9.6.8 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 9.14 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 dallas .
USERID	Remote user ID with which to execute the stop request. This must match the user ID originally used to start the component.
PWD	Password for the user ID.

Components

Universal Control

9.6.9 Stopping an HP NonStop Component via Universal Control

This example stops a component on a remote system.

```
run uctl -stop 10739132 -host dallas -userid joe -pwd akksdiq
```

Figure 9.15 Universal Control Manager for HP NonStop - Stop Component Example 1

Command Line Options

The command line options used in this example are:

Option	Description
-stop	ID of the component on the remote system to stop.
-host	Directs the command to a computer with a host name of dallas .
-userid	Remote user ID with which to execute the command. This must match the user ID originally used to start the component.
-pwd	Password for the user ID.

Components

Universal Control

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 Indesca 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 9.16, 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 9.16 UECLoad - List All Defined Brokers

9.7.2 Export a Specific Defined Broker

Figure 9.17, 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 9.17 UECLoad - Export a Specific Defined Broker

9.7.3 Export Events

Figure 9.18, 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 9.18 UECLoad - Export Events

9.7.4 Retrieve Archived File and Export

Figure 9.19, 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 9.19 UECLoad - Retrieve Archived File and Export

9.7.5 Delete a Specific Defined Broker

Figure 9.20, 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 9.20 UECLoad - Delete a Specific Defined Broker

9.7.6 Add Specific Defined Broker via deffile

Figure 9.21, 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 9.21 UECLoad - Add Specific Defined Broker via a Definition File

Figure 9.22, 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 9.22 UECLoad - Definition File used for Adding Specific Defined Broker

9.7.7 Add Existing Brokers to a Broker Group

Figure 9.23, 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 9.23 UECLoad - Add Existing Brokers to a Broker Group

9.7.8 Delete Existing Brokers to a Broker Group

Figure 9.24, 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 9.24 UECLoad - Delete Existing Brokers to a Broker Group

9.7.9 Export Events into ARC Format for z/OS

Figure 9.25, 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 9.25 UECLoad for z/OS - Export Events into ARC Format

9.7.10 Retrieve Archived File and Export into XML for z/OS

Figure 9.26, 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 9.26 UECLoad for z/OS- Retrieve Archived File and Export into XML

9.7.11 Export Events into ARC Format for Windows

Figure 9.27, 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 9.27 UECLoad for Windows - Export Events into ARC Format

9.7.12 Retrieve Archive File and Export into CSV for Windows

Figure 9.28, 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 9.28 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 chapter 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 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 <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.

HP NonStop Message Destinations

Message destinations are listed in the following table.

Destination	Description
STDERR	Messages are written to standard error.
LOGFILE	Messages are written to a log file. Not all programs provide this destination. Log files are written the \$SYSTEM.UNVLOG subvolume. All messages written to log files include a date and time stamp and the program's process ID.

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 Indesca 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	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

Indesca component error messages are translated, by the Universal Message Translator utility, into return (exit) codes based on a user-defined translation table.

Every command ends with a return code that indicates the success or failure of the command execution. Typically, a return code of 0 indicates success; all other codes indicate failure.

However, a small number of commands do not set their return code under failure conditions; instead, they issue error messages. Based on the user-defined translation table, Universal Message Translator translates these error messages into return codes.

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](#)

[Execute Universal Message Translator from z/OS Manager \(in a Script with Table Housed on Remote Server\)](#)

[Execute Universal Message Translator from z/OS Manager \(in a Script with Table Housed on z/OS\)](#)

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.

HP NonStop

Execute Universal Message Translator from HP NonStop

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 **-table** option points to the DD statement **TABLEDD**, which defines the return codes to end this process based on matching text. The first column defines the text to match; the second defines the return code to set if the matching text exists in the **SYSIN** DD.

The **-level** option turns on messaging. All messages will be written to **SYSPRINT**. The **SYSIN** DD statement points to the text file to be interrogated.

PARM Options

The PARM options used in this example are:

Option	Description
-table	Translation table file name.
-level	Level of messages that will be displayed.

Components

Universal Message Translator

11.3.4 Execute Universal Message Translator from z/OS Manager (in a Script with Table Housed on Remote Server)

Figure 11.7, below, illustrates the execution of Universal Message Translator from a z/OS Universal Command Manager.

```
//S1 EXEC UCMDPRC
//SCRIPTDD DD *
/opt/universal/ucmdsrv-2.2.0/bin/umet -file /home/log.file -table\
/home/umet.table -level verbose
/*
//SYSIN DD *
-host dallas
-script SCRIPTDD
-userid joe
-pwd abcdefg
/*
```

Figure 11.7 Universal Message Translator - Execute from z/OS Manager (with Table on Remote Server)

Universal Message Translator is executed in order to interrogate a log file and set the return code based on the translation table.

Since the command spans two lines, the native operating system continuation character must be used:

- / for UNIX
- ↵ for Windows

The full path to the Universal Message Translator executable must be specified for UNIX if the path is not part of the user’s profile.

Command Line Options

The command line options used in this example are:

Option	Description
-table	Translation table file name.
-level	Level of messages that will be displayed.
-file	Input message file name. If the option is not specified, UMET reads its input from stdin .

SYSIN Options

The SYSIN options used in this example are:

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

Components

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

[Universal Message Translator](#)

11.3.5 Execute Universal Message Translator from z/OS Manager (in a Script with Table Housed on z/OS)

Figure 11.8, below, illustrates the execution of Universal Message Translator from a z/OS Universal Command Manager.

The message table is stored and maintained on z/OS and copied down to the server upon execution. The `-table` option points to the table of defined return codes based on text. The `-file` option points to the text file to be interrogated.

```
//S1 EXEC UCMDPRC
//UNVIN DD DISP=SHR,DSN=h1q.umet.table
//SCRIPTDD DD *
UCOPY > c:\temp\umet.table
umet -table c:\temp\umet.table -file c:\temp\bkup.log -level verbose
/*
//SYSIN DD *
-host dallas
-script SCRIPTDD
-userid joe
-pwd abcdefg
/*
```

Figure 11.8 Universal Message Translator - Execute from z/OS Manager (with Table on z/OS)

The first command copies the messages table from the `UNVIN DD` of the manager process to a server file named `c:\temp\umet.table`. The `UMET` program is then executed to interrogate the log file and set the return code based on the translation table.

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, <code>UMET</code> reads its input from <code>stdin</code> .

SYSIN Options

The SYSIN options used in this example are:

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

Components

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

[Universal Message Translator](#)

11.3.6 Execute Universal Message Translator from Windows

Figure 11.9, below, illustrates the execution of Universal Message Translator from Windows.

```
umet -table c:\umettable.txt -file c:\umetfile.txt -level verbose
```

Figure 11.9 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
<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 <code>stdin</code> .

Components

Universal Message Translator

11.3.7 Execute Universal Message Translator from UNIX

Figure 11.10, 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/umetable.txt
-file /tmp/umetfile.txt -level verbose
```

Figure 11.10 Universal Message Translator - Execute from UNIX

The `-table` option points to the file, which defines the return codes with which to end this process, based on matching text.

The first column of the file defines the text to match; the second defines the return code to set if the matching text exists in the file defined by the `-file` option.

The `-level` option turns on messaging. All messages will be written to `stdout`.

Command Line Options

The command line options used in this example are:

Option	Description
<code>-table</code>	Translation table file name.
<code>-level</code>	Level of messages that will be displayed.
<code>-file</code>	Input message file name. If the option is not specified, UMET reads its input from <code>stdin</code> .

Components

Universal Message Translator

11.3.8 Execute Universal Message Translator from IBM i

Figure 11.11, 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 11.11 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 stdin , which is allocated to the terminal for interactive jobs and to QINLINE for non-interactive jobs.

Components

Universal Message Translator

11.3.9 Execute Universal Message Translator from HP NonStop

Figure 11.12, below, illustrates the execution of Universal Message Translator from HP NonStop.

```
run $SYSTEM.UNVBIN.umet -table umettable -file umetfile -level verbose
```

Figure 11.12 Universal Message Translator - Execute from HP NonStop

The parameter `-t` 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 `-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 <code>stdin</code> .

Components

Universal Message Translator

Monitoring and Alerting

The Monitoring and Alerting feature of Indesca provides for the monitoring the status and activity of all Indesca Agents in an enterprise and the posting of alerts regarding the statuses.

Monitoring is provided through continuous [Monitoring of All Agents](#) or by [Querying for Job Status and Activity](#) of a specific Agent.

12.1 Monitoring of All Agents

Indesca provides for the continuous monitoring of all Agents in an enterprise through its Universal Enterprise Controller component.

12.1.1 Monitored Information

Indesca monitors for three types of information:

1. Alerts for all Agents and SAP systems being monitored
2. Jobs (active, completed, and failed) for all Agents being monitored
3. Systems (Agents and SAP systems) being monitored

This information can be viewed via the I-Activity Monitor client application.

12.1.2 Polling

Indesca periodically polls each Agent and SAP system in an enterprise in order to retrieve its status information.

It determines whether or not a change in status of the Agent or SAP system has occurred since the last poll. If the status has changed, it sends this information to the I-Activity Monitor.

12.1.3 Alerts

Indesca sends out alerts to any connected Agent-monitoring applications whenever:

- Agent is unreachable.
- Agent is not responding.
- Agent component enters an orphaned or disconnected state.

These alerts are posted to the:

- Event Log (when running under Windows)
- Console (when running under z/OS)

Automation tools can be used in conjunction with these messages to perform operations based on agent failures.

12.2 Querying for Job Status and Activity

Indesca has the ability to query any specific Universal Broker in an enterprise for Broker-related, and active component-related, activity via the Universal Query utility.

Universal Query returns information for a Universal Broker that is installed on the host, as specified by configuration options on the command line or in a configuration file. Information regarding the components managed by a particular 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.

12.3 Querying for Job Status and Activity Examples

This chapter provides user scenarios for the Querying Job Status feature of Indesca.

Links to detailed technical information on appropriate Indesca 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](#)

HP NonStop

[Universal Query for HP NonStop](#)

12.3.1 Universal Query Output

Figure 12.1, below, illustrates an example of the output generated by the execution of the Universal Query utility.

This sample output is from the execution of Universal Query to host `dallas.domain.com` using a NORMAL report.

```
Universal Query Report
for
Mon 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

12.3.2 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 dallas .

Components

Universal Query

12.3.3 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

12.3.4 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

12.3.5 Universal Query for HP NonStop

The Universal Query utility is used to list all active components on a remote server. The output will be written to stdout.

```
run $SYSTEM.UNVBIN.uquery -host localhost
```

Figure 12.6 Universal Query for HP NonStop - 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

Windows Event Log Dump

13.1 Overview

Indesca provides the ability to select records from a Windows event log and write them to a specified output file via its Universal Event Log Dump utility.

All records from a log can be dumped, or event records can be selected according to the date and time that they were generated.

Universal Event Log Dump can be run any time as a stand-alone application. It also is designed to work with Universal Command, which provides centralized control from any operating system and additional options for redirecting output.

Universal Event Log Dump consists of the command line program (`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
-type	Event log to be dumped.
-stime	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 Indesca 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. The Universal Broker processes and responds to all database access requests it receives from individual Indesca components.

14.2 Component Information Database

The component information database records information about all Indesca 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 Indesca server component. Each time a component's state changes, it sends a notification to Universal Broker, which updates that component's record.

For the Indesca components that offer it, this component state provides the basis for reconnect or restart functionality, otherwise known as network or manager fault tolerance, respectively (see Chapter 15 [Fault Tolerance Implementation](#)).

This state serves as the basis for support of reconnect or restart functionality; that is, fault tolerance (see Chapter 15 [Fault Tolerance Implementation](#)).

When an Indesca server process finishes executing and its component state indicates that it has completed, Universal Broker deletes that component's information from the database.

Universal Broker stores Indesca 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

Indesca 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`.

14.2.1 Universal Spool

The Universal Spool – or more simply, the spool – supports storage of the standard file I/O that Universal Command Server redirects from user processes. When Universal Command Server executes with spooling enabled, it captures a user process' standard input, standard output, and standard error and writes them to the spool.

Universal Spool is implemented as a set of databases. Universal Broker and Universal Command Server remove the database records automatically when they are no longer required. No database maintenance jobs are required.

Universal Broker and server components are the only programs that access the spool. No user access is required. The operating system's file system security should be used to prevent all access to the spool except for the broker and server. The Broker and server require full control permissions to the spool in order to add, delete, update, and read database files.

All standard I/O files written to the spool are encrypted to insure data privacy.

The spool database files that store redirected standard input, standard output, and standard error have the format `spool.stdin.compид.db`, `spool.stdout.compид.db`, and `spool.stderr.compид.db`, respectively. In each of the names, `compид` is replaced by the server's actual component ID.

Windows

The database files' default location is:

- `C:\Program Files\Universal\spool\ucmdsrv` (32-bit Windows systems)
- `C:\Program Files (x86)\Universal\spool\ucmdsrv` (64-bit Windows systems)

UNIX

These files reside in the `/var/opt/universal/spool` directory.

z/OS

Indesca components access these files 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

These files are located in the spool library, `UNVSPL420`, in the following format:

- `SExxxxxxx` (SE = standard error)
- `SOxxxxxxx` (SO = standard output)
- `SIxxxxxxx` (SI = standard input)

In each file name, `xxxxxxx` is the component ID expressed in hexadecimal. (The component ID is the Component ID used to identify a Stonebranch process. This is the same Component ID output by Universal Query.)

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](#)

These 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 systems)

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 Reference Guide](#).



Stoneman's Tip

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 `comp_info_retention` option to a value greater than 0 causes the event spool record to be preserved.

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. [Customer Support](#).

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.

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 [6.5 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 4.2.0 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 Indesca Database Records Examples

This section contains examples demonstrating the listing of Indesca database records via the Universal Spool List (USLIST) utility.

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

Windows and UNIX

[List Universal Broker Database](#)

[List Universal Command Server 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 Command Server Database Records

Figure 14.3 and Figure 14.4, below, illustrate how to list the database records for a specific component (in this case, Universal Command Server).

Windows

```
cd c:\program files\universal\uspool\bin
uslist -list ucmd -ucmdspooldir "c:\program files\universal\spool2"
```

Figure 14.3 Universal Spool List for Windows - List Universal Server Database Records

UNIX

```
cd /opt/universal/bin
uslist -list ucmd -ucmdspooldir "c:\program files\universal\spool2"
```

Figure 14.4 Universal Spool List for UNIX - List Universal Server Database Records

These examples list the contents of the Universal Command Server Component database. A summary of all records is written.

These examples specify the directory location in which the Universal Command Server Component database is located. If the directory contains spaces, it must be enclosed in double (") quotes.

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 (scomponent.db) 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
-component	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 Indesca Database Records

This section contains examples demonstrating the removal of Indesca database records via the Universal Spool Remove utility.

Links to detailed technical information on appropriate Indesca 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
-component	Component identifier for which records will be removed.
-brokerspooldir	Directory location in which the Universal Broker Component database (bcomponent.db) is located

Components

Universal Spool Remove

Fault Tolerance Implementation

15.1 Overview

For Indesca, fault tolerance is the capabilities 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, fault tolerance – for both network and component managers – is implemented in two Indesca components:

- Universal Command
- Universal Connector

15.2 Network Fault Tolerance in Universal Command

Universal Command uses the TCP/IP protocol for communications over a data network. The TCP/IP protocol is a mature, robust protocol capable of resending packets and rerouting packets when network errors occur. However, data networks do have problems significant enough to prevent the TCP/IP protocol from recovering. As a result, the TCP/IP protocol terminates the connection between the application programs.

As with any application using TCP/IP, Universal Command is subject to these network errors. Should they occur, a product can no longer communicate and must shut down or restart. These types of errors normally show themselves as premature closes, connection resets, time-outs, or broken pipe errors.

15.2.1 Network Fault Tolerant Protocol

Universal Command provides the ability to circumvent these types of errors with its Network Fault Tolerant protocol. By using this protocol, Universal Command traps the connection termination caused by the network error and reestablishes the network connections. When connections have been reestablished, processing resumes automatically from the location of the last successful message exchange. No program restarts are required and no data is lost.

The Network Fault Tolerant protocol acknowledges successfully received messages and checkpoints successfully sent messages. This reduces data throughput. Consequentially, the use of network fault tolerance should be weighed carefully in terms of increased execution time versus the probability of network errors and cost of such errors. For example, it may be easier to restart a program than to incur increased execution time.

When a network connection terminates, the manager enters a network reconnect phase. In this phase, the manager attempts to connect to the server and reestablish its network connections. The condition that caused the network error can persist only for seconds, or it can persist for days.

The manager attempts server reconnection for a limited amount of time, as specified by the following configuration options:

- `RECONNECT_RETRY_COUNT` (number of retry attempts)
- `RECONNECT_RETRY_INTERVAL` (frequency of retry attempts)

If all attempts fail, the manager ends with an error.

When a network connection terminates, the server's action depends on whether or not it is executing with manager fault tolerance.

Without manager fault tolerance, the server enters a disconnected state and waits for the manager to reconnect. The user process continues running. However, if the user process attempts any I/O on the standard files, it will block. The server waits for the manager to reconnect for a period of time defined by the manager's `RECONNECT_RETRY_COUNT` and `RECONNECT_RETRY_INTERVAL`. When that time has expired, the server terminates the user process and exits.

With manager fault tolerance, the server continues executing in a disconnected state. The server satisfies all user process standard I/O requests. The user process does not block. It continues to execute normally. When the user process ends, the server waits for a manager reconnect for a period of time defined by the `JOB_RETENTION` option.

15.2.2 Universal Command Manager

Universal Command Manager starting with version 2.1.0 can request the use of the network fault tolerant protocol. If the server does not support the protocol or is not configured to accept the protocol, the manager continues without using the protocol.

The `NETWORK_FAULT_TOLERANT` option is used to request the protocol.

15.2.3 Universal Command Server

Universal Command Server starting with version 2.1.0 can be configured with or without the network fault tolerant protocol. The `NETWORK_FAULT_TOLERANT` option is used to configure the network fault tolerant protocol.

If the server is configured with network fault tolerance off, the manager cannot override it. If the server is configured with the network fault tolerance on, the manager option determines if the protocol is actually used or not.

15.3 Manager Fault Tolerance in Universal Command

Distributed applications are comprised of many independent components running on host systems, throughout the enterprise, connected with a data network. Many of the host systems are in different physical locations, in different organization groups, and have different system management policies. It can be difficult to schedule individual host downtime when there are so many overlapping requirements.

Host systems must be shut down at scheduled intervals and, unfortunately, at unscheduled intervals. The impact of a system being down must be minimized by a distributed application.

With the Manager Fault Tolerant feature, Universal Command components can be shut down and restarted at a later time. After a manager has been started, it can be terminated and restarted at a later time. It can be shut down for any period of time: seconds, days, or even months. The server can be shut down as well. When it is restarted, any work that was completed prior to the shutdown is immediately available for a manager to retrieve.

15.3.1 Functionality

The basic functionality of manager fault tolerance is:

1. Manager requests the execution of a command on a remote system.
2. Command executes on the remote system, optionally reading and writing data.
3. Manager redirects:
 - Its standard input data to the standard input of the remote command.
 - Standard output file and standard error file of the remote command to its own standard output and standard error.

If the manager is terminated or the manager's host system is shut down, the remote command cannot read the manager's standard input or write its standard output and error files. Without manager fault tolerance, the remote command must terminate, since its data source and destination are now gone. Otherwise, it would wait forever.

Manager fault tolerance provides an execution environment in which the manager is not required in order for the user process to continue execution on the remote system. The user process can execute to completion with or without a manager connected.

When the manager starts a user process, the manager executes as normal; standard output and standard error files are redirected back to the manager as the user process produces the data. The difference is data spooling. In order for the user process to have real-time access to its input and output, the data is spooled in the Universal Spool Database. The spool provides complete independence from the manager. The spool subsystem satisfies all data requirements for the user process via the Universal Command Server.

The manager can terminate and a new manager can restart and reconnect to the user process. If the user process has completed, the new manager receives the user processes standard files and its exit status. The restarted manager behaves in all ways as if it was the originating manager.

Command Identifier

A manager requests manager fault tolerance with the `MANAGER_FAULT_TOLERANT` option and by providing a command identifier (command ID) using the `COMMAND_ID` option. The command ID identifies the unit of work being executed. In this context, a unit of work includes the manager, server, and user process.

The manager indicates to the server that this request is restartable. The `COMMAND_ID` option provides a command identifier that uniquely identifies the server and user process on the remote host. When a manager is restarted, it must provide the same command ID identifying the server and user process with which it wants to reconnect.

Providing a unique command ID is not trivial. Many managers may be executing on many different hosts, and all executing work on the same server host. It is possible for a manager to start a restartable command from one host, terminate, and restart on a completely different host.

The command ID value can be any text value of unrestricted length. In practical terms, the character set and limits on command line length of the manager host impose restrictions on the value.

Standard I/O Files

The Universal Spool system satisfies all user process data requests via the Universal Command Server. When the user process reads from its standard input file, the server reads it from the spool and provides it to the user process. When the user process writes to standard output or error, the server receives the data and writes it to the spool.

A manager requesting restart capability (manager fault tolerance) first transfers its entire standard input file to the server, which it in turns writes to the spool. When all data has been received, the server creates the user process. This provides complete manager independence for the entire life of the user process.

As long as the manager is connected, the standard output and standard error files are transferred to the manager, as the user process produces the data, all in real-time. The data also is written to the spool. If the manager terminates, the data is written to the spool only.

A restarted manager is sent all of the standard output and standard error files, from the beginning, that currently is spooled. If the user process still is executing, the restarted manager will receive all of the data currently spooled. When it has caught up with the data being produced, the manager starts to receive the data from the user process as it is written.

Requesting Restart

When a restartable manager is initiated, it is either an initial instance or a restarted instance of a command ID. The command ID identifies a unit of work represented by the manager, server, and user process. See [Command Identifier](#), above, for more information on the command ID.

The [RESTART](#) option specifies whether or not the manager instance is requesting a restart of a previous command ID. Possible [RESTART](#) values are **yes**, **no**, and **auto**.

The **auto** value specifies:

- If there is no existing command ID executing on the remote host, consider this manager execution the first instance.
- If there is an existing command ID, and it is not connected to any manager, consider this a restart of the command ID.

The **auto** value permits automatic restart by eliminating the need to modify the [RESTART](#) value for the initial instance and restarted instance.

Note: The **auto** value cannot be used with a [COMMAND_ID](#) value of *, which specifies that the UCMD Manager will generate a unique command ID for each run.

Case Example 1: Normal Execution

The following figure diagrams the sequence of events that occur when a restartable manager requests the execution of a command on a remote host. In this case the manager and server remain executing and connected until normal completion of the user process.

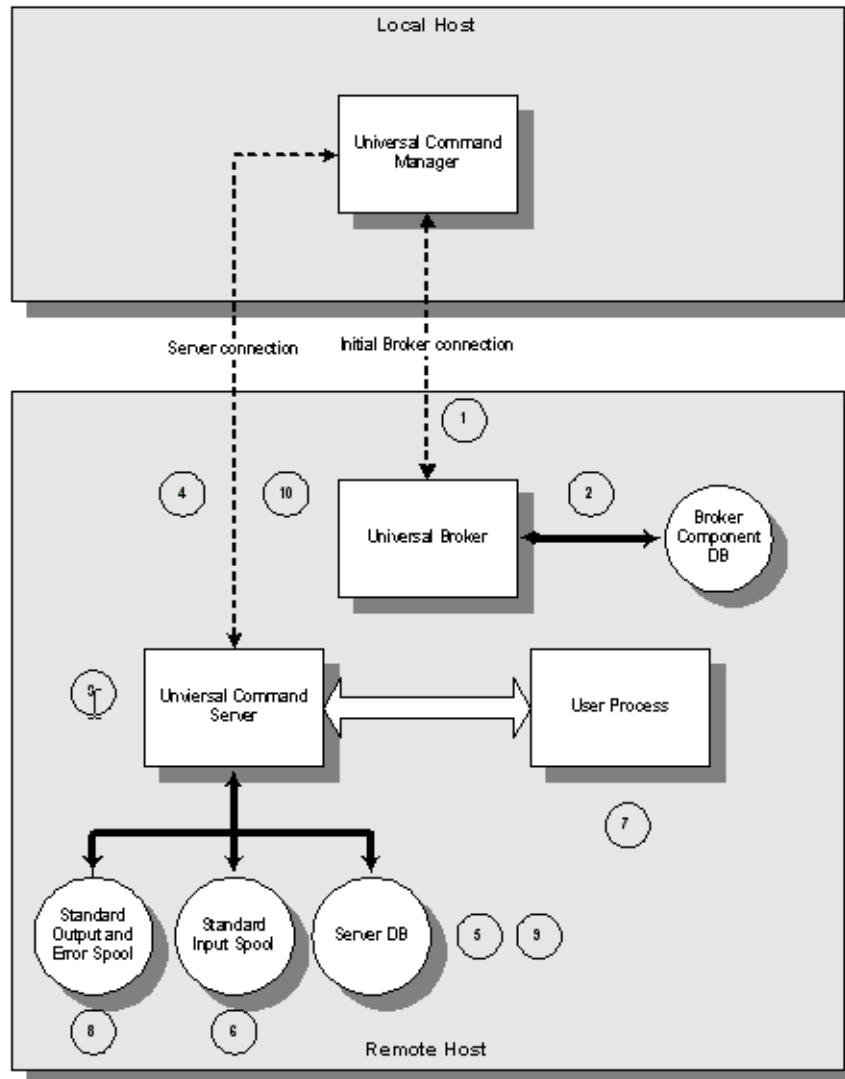


Figure 15.1 Case Example 1

The Local Host is the host on which the manager is being executed. The Remote Host is the host on which the manager is requesting command execution.

The components involved are:

- **Universal Command Manager**
The manager requests remote execution of a command or script. The manager executes the remote command in a manner such that the command appears to be executed locally.
- **Universal Broker**
The broker manages Universal component execution.
- **Universal Command Server**
The server executes the manager requested command and processes the user process's standard I/O requests.
- **User Process**
The user process represents the manager requested command.

The diagram demonstrates the sequence of events that occur when a restartable manager requests command execution on a remote host. The numbers enclosed in circles represent the sequence of events and correspond to the listed descriptions below.

1. The manager connects to the broker and sends a request to start a Universal Command Server. The start request from the manager requests manager fault tolerance and includes the command ID to identify the unit of work.
2. The broker records the unit of work in the Broker Component Database as restartable for possible future restarts.
3. The broker starts an instance of the Universal Command Server.
4. The manager and server exchange messages that specify all options used to carry out the request.
5. The server records the unit of work in the Universal Command Server Database for possible future restarts.
6. The manager sends all standard input data to the server and the server writes the standard input data to the Universal Spool database.
7. Once all standard input is spooled, the server starts the user process.
8. As the user process writes standard output and standard error data, the server writes the data to the Universal Spool database. If the manager is connected to the server, the data is written to the manager as well.
9. The user process executes until completion. Once the user process completes, the server writes the exit status of the user process to the Universal Command Server Database.
10. The server sends the exit status to the manager. This completes the unit of work.

Case Example 2: Restart when User Process is Executing

The following figure diagrams the sequence of events that occur when a manager requests a restart of a currently executing unit of work. In this case the initial instance of the manager terminated. A restarted instance of the manager is started and requests to be reconnected to the unit of work.

This example continues from the first case example. Please refer to first case example for details of the component descriptions included in the diagram.

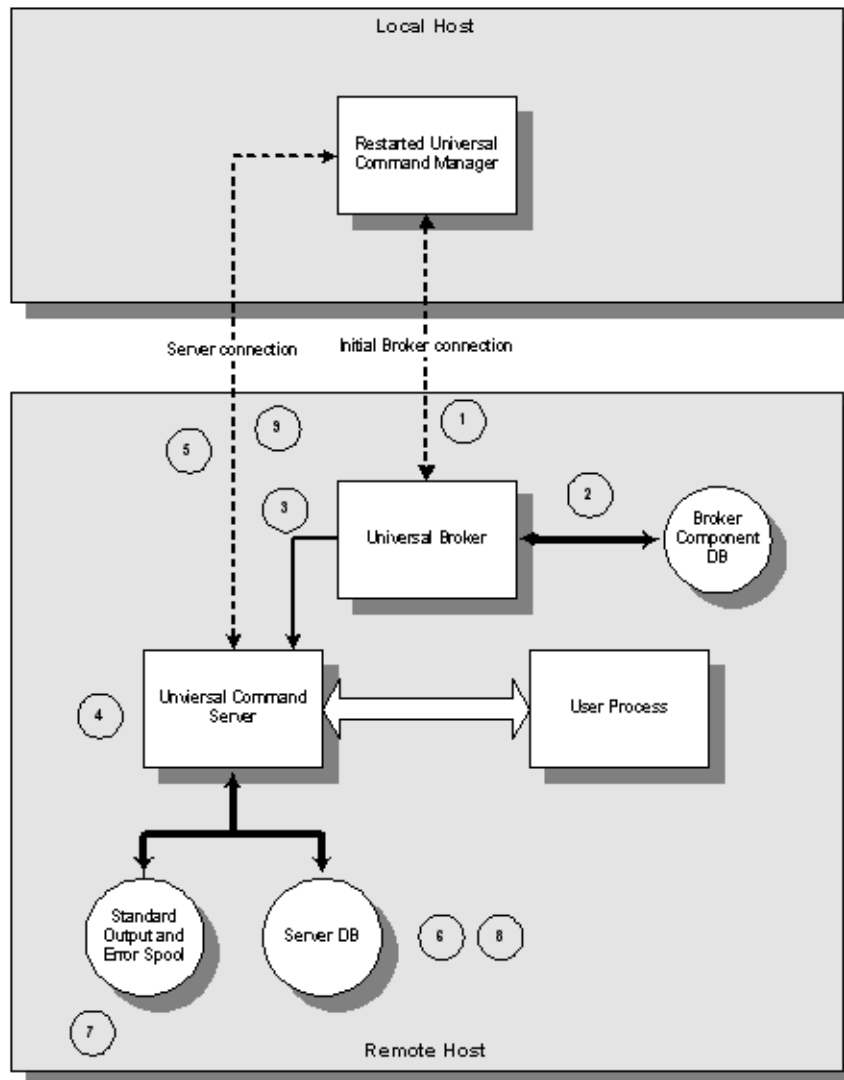


Figure 15.2 Case Example 2

The diagram demonstrates the sequence of events that occur when a manager requests to be restarted with a unit of work identified by a command ID. The numbers enclosed in circles represent the sequence of events and correspond to the listed descriptions below.

1. The restarted instance of the manager sends a restart request to the broker. The restart request contains the command ID specified as part of the invocation of the manager.
2. The broker verifies that the component is restartable and that the components communication state is acceptable for a restart request. If the server component were currently connected to a manager, its communication state would not permit a restart request.
3. The broker sends the restart request to the server corresponding to the command ID.
4. The server authenticates the request with the manager-supplied user ID and password. The password must be the same as the initial manager instance.
5. The manager and server exchange options that are used to carry out the request.
6. The server records the restart in the Universal Command Server Database.
7. The server sends spooled standard output and error files to the manager. This is performed while the user process may still be writing standard output and error to the spool. Once all spooled output is sent to the manager, the server will send standard output and error from the user process as it is being produced.
8. The user process executes to completion. The server records the user process exit status in the Universal Command Server Database.
9. The server sends the exit status to the manager. This completes the unit of work.

Case Example 3: Restart when User Process has Ended

The following figure diagrams the sequence of events that occur when a manager requests a restart of a unit of work that has completed. In this case the initial instance of the manager has terminated, the user process completed normally, and a restarted instance of the manager is started and requests to be reconnected to the completed unit of work.

This example continues from the first case example. Please refer to first case example for details of the component descriptions included in the diagram.

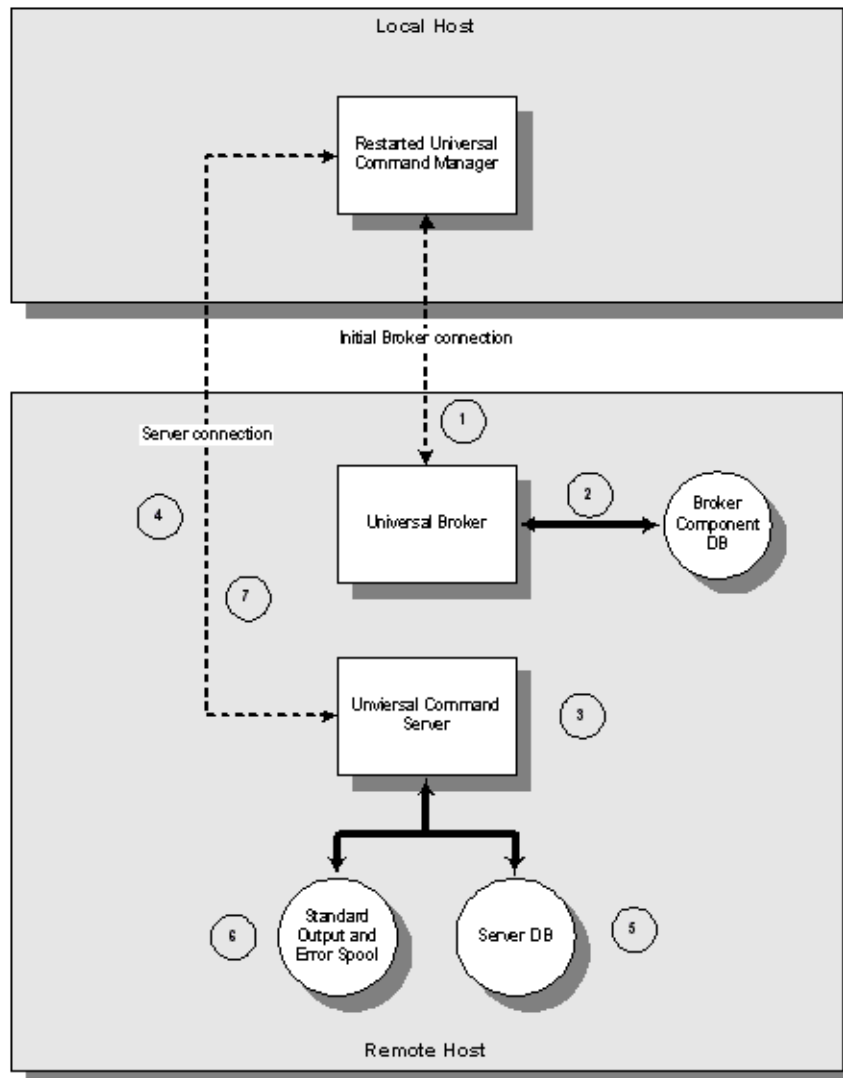


Figure 15.3 Case Example 3

The diagram demonstrates the sequence of events that occur when a manager requests to be restarted with a unit of work identified by a command ID. The user process in this case has completed execution. The numbers enclosed in circles represent the sequence of events and correspond to the following descriptions:

1. The restarted instance of the manager sends a restart request to the broker. The restart request contains the command ID specified as part of the invocation of the manager.
2. The broker verifies that the component is restartable and that the components communication state is acceptable for a restart request. If the server component were currently connected to a manager, its communication state would not permit a restart request.
3. Since the user process has completed, the broker starts a new server to process the restart request. The server authenticates the request with the manager-supplied user ID and password. The password must be the same as the initial manager instance.
4. The manager and server exchange options that are used to carry out the request.
5. The server records the restart in the Universal Command Server Database.
6. The server sends spooled standard output and error files to the manager.
7. The server sends the user process exit status to the manager. This completes the unit of work.

15.3.2 Component Management

In order to fully understand Universal Command fault tolerant features, some understanding of how the Universal Broker manages components is necessary.

Universal Broker manages component startup, execution, and termination. The broker and its components have the ability to communicate service requests and status information between each other.

The Broker maintains a database of components that are active or have completed and waiting for restart or reconnection. The component information maintained by the broker determines the current state of the component. This state information is required by the broker to determine if a restart or reconnect request from a manager is acceptable or not. The broker's component information can be viewed with the Universal Query program.

One piece of component information maintained by the broker is the component's communication state. The communication state primarily determines what state the Universal Command Server is in regarding its network connection with a manager and the completion of the user process and its associated spooled data.

Table 15.1, below, describes the communication state values.

- **Reconnect** column indicates whether or not a network reconnect request is valid.
- **Restart** column indicates whether or not a restart request is valid.

State	Reconnect	Restart	Description
STARTED	NO	NO	Server has started. If the server is restartable it is receiving the standard input file from the manager and spooling it.
ESTABLISHED	NO	NO	Server and manager are connected and processing normally. This state is the most common state when all is well.
DISCONNECTED	YES	YES	Server is not connected to the manager. This occurs when a network error has occurred, the manager halted, or the manager host halted. The server is executing with either the network fault tolerant protocol, is restartable, or both. Note: The server cannot tell if the manager is still executing or not since it cannot communicate with it.
ORPHANED	NO	YES	Manager has terminated. The manager sends a termination message to the server to notify it of its termination prior to terminating. This state only occurs if the server is restartable.
RECONNECTING	NO	NO	Server has received a reconnect request from the manager to recover a lost network connection. This state should not remain long, only for the time it takes to re-establish the network connections.

State	Reconnect	Restart	Description
RESTARTING	NO	NO	Server has received a restart request from the manager. This state should not remain long, only for the time it takes to re-establish network connections.
PENDING	NO	YES	A restartable server and its user process have completed. The user process standard output and error files are in the spool. A manager has not been restarted to pick up the spooled files and user process exit status. The server remains in this state until a manager is restarted.
COMPLETED	NO	NO	Server and manager have completed. All standard output and standard error files have been sent to the manager and the user process's exit status.

Table 15.1 Component Communication States

15.4 Network Fault Tolerance in Universal Connector

Universal Connector commands are processed by calling appropriate BAPI functions in the SAP system. The BAPI function calls are issued over an RFC connection. Universal Connector provides fault tolerance at the RFC level. If an RFC call fails, that call is retried until it completes successfully, or exceeds a user definable retry limit.

If an RFC call fails, Universal Connector will close the current RFC connection and establish a new RFC connection in order to continue processing. The process of establishing and preparing an RFC connection is referred to in this document as the RFC logon process. The RFC logon process involves establishing an RFC connection, logging on to the XMI interface and setting the XMI audit level. If the RFC logon process fails, it will be retried until it completes successfully, or exceeds a user definable retry limit. When the new RFC connection is successfully established, Universal Connector will reissue the failed RFC call.

The entire process of establishing a new RFC session and reissuing the failed RFC call will be retried until either

- RFC call completes successfully.
- User-definable RFC retry limit is exceeded.

Certain BAPI functions should not be retried in an unknown state. Those BAPI functions are points of failure within the Universal Connector fault tolerant solution. Section [15.4.1 Points of Failure](#) lists the points of failure and their relationship to Universal Connector commands.

15.4.1 Points of Failure

The points of failure within Universal Connector fault tolerant architecture are:

- **Job Submission**
Some BAPI functions called in the submission process cannot be restarted in an unknown state without possible negative consequences. If an RFC call fails issuing those BAPI's, Universal Connector will end unsuccessfully.
- **Job Modification**
Some BAPI functions called in the job modification process can not be restarted in an unknown state without possible negative consequences. If an RFC call fails issuing those BAPI's, Universal Connector will end unsuccessfully.
- **Job Start**
Some BAPI functions called in the job start process can not be restarted in an unknown state. If an RFC call fails issuing those BAPI's, Universal Connector will end unsuccessfully.

15.4.2 Network Fault Tolerance Configuration Parameters

The set of Universal Connector configuration parameters that can be used to fine-tune the fault tolerance support for a particular environment are:

- RFC Logon Retry Count
- RFC Logon Retry Timeout
- RFC Timeout
- RFC Retry Count
- RFC Retry interval
- RFC Listen Interval

See Sections [2.4.10 RFC Options](#) and [3.4.11 RFC Options](#) in the [Universal Connector Reference Guide](#) for details concerning the use of these parameters.

15.5 Client Fault Tolerance - Universal Connector

The Client Fault Tolerance feature allows the Universal Connector client application to be shut down and restarted at a later time. This functionality helps avoid problems that can result if the Universal Connector application terminates unexpectedly while processing an SAP job. In such an instance, the Client Fault Tolerance restart capability allows Universal Connector to reconnect to a running (or completed) job while preventing the unintentional start of a new instance of the original SAP job.

15.5.1 Overview

To achieve Client Fault Tolerance, Universal Connector must be able to associate the SAP jobs it defines and starts with a particular unit of work. In this context, a unit of work includes the Universal Connector client and SAP job instance.

To associate an SAP job with a particular unit of work, the user must be able to specify some identifying characteristic that is specific to that unit of work. The SAP system uniquely identifies job instances by a job name/job ID pair. Since the job name must be reusable and the job ID is assigned by the SAP system at the time of definition, Universal Connector must use an alternative job characteristic. This alternative job characteristic is the Universal Connector command Identifier.

Universal Connector references a particular unit of work by a job name/Command Identifier combination. The Universal Connector command identifier is tied to the SAP job by appending a Command ID Job Step to the SAP job associated with the Universal Connector command instance. The Command ID job step is required for identification purposes only. Therefore, the program used for the Command ID step is intended to add minimum overhead to the job. The Command ID used for the job is included in the definition of the Command ID job step.

15.5.2 Modes

Universal Connector supports two modes of client fault tolerance:

1. Pre-XBP 2.0 Client Fault Tolerance (CFT)
2. Secure Client Fault Tolerance (Secure CFT)

Pre-XBP 2.0 Client Fault Tolerance (CFT)

This mode is the original implementation of client fault tolerance used prior to the release of XBP 2.0. Due to limitations in the XBP 1.0 interface, Universal Connector client fault tolerance on pre-XBP 2.0 SAP systems uses an external program step as the command ID job step. Using an external program step as the command ID job step has the following security and ease of use drawbacks:

- **Security drawback**
Using an external program job step requires the SAP user ID to have authority to run external programs. This authority cannot be given lightly for the following reason: When running an external job step, the SAP system first performs an authorization check to see if the user ID has the right to run an external program. If so, the external program is run under the user ID of the user who started the SAP system
- **Ease of use drawback**
Using an external program job step requires a target host be specified for the external program to run on. This requires information about the SAP landscape that may not be readily available. Also, this presents the possibility of the Universal Connector job's parameters becoming out of sync with the SAP landscape.

Secure Client Fault Tolerance (Secure CFT)

This mode is an enhancement of the original implementation. The secure CFT mode requires XBP 2.0 to be installed on the SAP side of the Universal Connector connection. In this mode, an ABAP program step is used for the command ID job step. Using an ABAP program step as the Command ID job step eliminates the security and ease of use drawbacks mentioned above for external program job steps.

- **Security**
The execution of ABAP programs and the resources required by them are secured by SAP authorization checks.
- **Ease of Use**
ABAP program job steps do not require a target host. They run on whichever application server the job runs on. Therefore, there are no target specific parameters required for secure CFT mode.

The mode of client fault tolerance to be used by Universal Connector is determined by the value of the `SECURE_CFT` option. Valid values for this option are **yes** and **no**:

- **yes** will cause USAP to use secure CFT mode.
- **no** will cause USAP to use the original pre-XBP 2.0 mode of client fault tolerance.

The default value is **yes**.

Both modes of CFT follow the same basic process flow. When Universal Connector is requested to restart a particular command ID job, it queries the SAP system for all jobs with the specified job name. The list of jobs returned by the SAP system is scanned for a job that contains an appropriate Command ID Job Step. If found, Universal Connector will re-connect to the SAP job instance and satisfy the command line requirements.

Universal Connector is capable of restarting a command ID as long as the associated command ID job remains in the SAP system.

15.5.3 Parameters

Client Fault Tolerance Target Host

The client fault tolerance target host parameter is only required for pre-XBP 2.0 client fault tolerance mode. If the secure CFT mode is being used, the client fault tolerance target host parameter is ignored.

As part of an external program command ID job step definition, SAP requires a target host on which to run the external program (echo). Universal Connector provides the client fault tolerance target host parameter to specify the target host for the command ID job step.

The Client Fault Tolerance Target Host is specified with the [CFT_TARGET_HOST](#) option.

Client Fault Tolerance Command Prefix

The client fault tolerance command prefix parameter is only required for pre-XBP 2.0 client fault tolerance mode. If the secure CFT mode is being used, the client fault tolerance command prefix parameter is ignored.

The external program command ID job step has the potential to be run on any operating system reachable by the SAP system. The operating system that the Command ID Job Step runs on is that which exists on the host system specified by the Client Fault Tolerance Target Host parameter. Different operating systems may require commands to be called in different ways. Therefore, the Client Fault Tolerance Command Prefix parameter allows the user to specify the prefix necessary to run the echo command on the host system specified by the Client Fault Tolerance Target Host parameter.

For example, to run the echo command on a Windows system, the following command line would be required for an SAP external job step: `cmd /C echo`. Therefore, the Client Fault Tolerance Command Prefix for this system would be: `cmd /C`.

The Client Fault Tolerance Command Prefix parameter is specified with the [CFT_COMMAND_PREFIX](#) option.

Secure Client Fault Tolerance Option

The mode of client fault tolerance to be used by USAP is determined by the value of the [SECURE_CFT](#) option. Valid values for this option are **yes** and **no**:

- **yes** will cause Universal Connector to use secure CFT mode.
- **no** will cause Universal Connector to use the original pre-XBP 2.0 mode of client fault tolerance.

The default value is **yes**.

Client Fault Tolerance ABAP Program

The client fault tolerance ABAP program parameter is only required for secure CFT mode. If the pre-XBP 2.0 CFT mode is being used, the client fault tolerance ABAP program parameter is ignored.

The client fault tolerance ABAP program parameter is used to specify the ABAP program to use for the command ID job step. Any ABAP program can be specified. The Universal Connector internal default ABAP program to use is BTCTEST. BTCTEST is a standard SAP ABAP program that should be available on all SAP systems. It does no real processing and can be considered a dummy program that does not interfere with job processing and places little overhead on the system.

15.5.4 Command ID Job Step

USAP creates Command ID jobs by appending a job step to the user's SAP job being defined to the system. This appended job step is the Universal Connector Command ID Job Step.

Pre-XBP 2.0 CFT Mode

In pre-XBP 2.0 CFT mode, the Universal Connector Command ID Job Step executes the external program `echo`. A string containing the command ID is inserted in the parameter field of the job step. The `echo` command is lightweight, does not interfere with the original job, and results in the command ID being printed to the joblog.

Secure CFT Mode

In secure CFT mode, the USAP command ID job step executes an ABAP program. The ABAP program defined to the command id step is user configurable with the `covetable` parameter. Any ABAP program can be specified. The Universal Connector internal default ABAP program to use is `BTCTEST`. `BTCTEST` is a standard SAP ABAP program that should be available on all SAP systems. It does no real processing and can be considered a dummy program that does not interfere with job processing and places little overhead on the system.

15.5.5 Command Identifier

Universal Connector requests client fault tolerance by providing a command identifier. The command identifier is specified on the command line with parameter `-cmdid`. The command ID/job name pair identifies the unit of work being executed.

The command ID option provides a command identifier that (paired with job name) uniquely identifies the SAP job on the SAP system. When a Universal Connector job is restarted, it must provide the same command ID identifying the SAP job with which it wants to reconnect.

Providing a unique command ID is not trivial. Many Universal Connector clients may be executing on many different hosts, all executing work on the same SAP system. It is possible for a Universal Connector client to start a restartable job from one host, terminate, and restart on a completely different host.

The command ID value can be any text value up to 245 characters in length. In practical terms, the character set and limits on command line length of the Universal Connector host may impose further restrictions on the value.

15.5.6 Requesting Restart

When a restartable Universal Connector command is initiated, it is either an initial instance or a restarted instance of a command ID. The RESTART option is specified on the command line with parameter **-restart**. The RESTART option specifies whether the Universal Connector command instance is requesting a restart of a previous command ID or not. Possible RESTART values are **yes**, **no**, or **auto**. The **auto** value specifies that if there is no existing command ID job on the SAP system, consider this Universal Connector execution the first instance. If there is an existing command ID job, consider this a restart of the command ID. The **auto** value permits automatic restart by eliminating the need to modify the RESTART value for the initial instance and restarted instance.

It is important to note that when using the RESTART **auto** value, Universal Connector will not start a new instance of a job on the SAP system if a job matching the job name/command ID exists in the SAP system. Universal Connector will continue to reconnect to the existing SAP job. Without considering the behavior resulting from the use of RESTART **auto**, it may be possible for one to assume that a job has been run multiple times when, in fact, Universal Connector has been reconnecting to the same job instance. Informational messages are printed by Universal Connector to standard error to indicate the reconnected status but, if the message level is not set to **info**, the messages will not be seen.

Given the possibility for confusion surrounding the use of RESTART **auto**, a parameter has been introduced to control the use of RESTART **auto**. This parameter is described in the following section.

Controlling Auto Restart

If not properly understood, the auto restart behavior described in the previous section can have potentially serious consequences. For this reason, a parameter has been introduced that will allow or disallow the use of auto restart. The parameter is available at the configuration file level and can be overridden at the command line level.

Configuration File Parameter

The configuration file parameter is **auto_restart_ok**.

Valid values for **auto_restart_ok** are **yes** and **no**:

- **yes** allows the use of auto restart.
- **no** prevents the use of auto restart.

Command File Parameter

The command line parameter **-autorestartok** can be used to override the configuration file setting.

Valid values for **-autorestartok** are **yes** and **no**:

- **yes** allows the use of auto restart
- **no** prevents the use of auto restart.

15.5.7 Sample Command Lines For Working With Client Fault Tolerance

Working With Job Definition Files

Initial Run of a Command ID Job

The following example will submit, start, and wait for the command ID job defined in job definition file `jobdef`. Because the `RESTART` option is set to `no`, Universal Connector will scan the SAP system to ensure that a command ID job with the same job name/command ID pair does not already exist on the system.

If a matching command ID job is found on the SAP system, Universal Connector will exit with an error code before performing the job submission.

Pre-XBP 2.0 CFT Mode

```
usap -user bob -pwd secret -submit jobdef -start -wait -cft_secure_cft no  
-cft_target_host pwndf0196 -cft_cmd_prefix "cmd /C" -cmdid 0000000001 -restart no
```

Note: The Client Fault Tolerance Command ID Prefix is set up for a Windows host. In many user environments, the Client Fault Tolerance Command ID Prefix parameter can be specified in the configuration file and will never need to be specified on the command line. The same may be true for the Client Fault Tolerance Target Host parameter. The secure CFT option would also be set in the configuration file in most cases.

Secure CFT Mode

```
usap -user bob -pwd secret -submit jobdef -start -wait -cft_secure_cft yes  
-cft_abap BTCTEST -cmdid 0000000001 -restart no
```

Note that in secure CFT mode, the `cft_secure_cft` and `cft_abap` parameters would most likely be specified in the Universal Connector configuration file.

Restart of a Command ID Job

In the following example, USAP is requested to restart command ID job 0000000001. Universal Connector will first parse the `jobdef` file to determine the jobname, and then scan the SAP system for a matching command ID job.

If a matching command ID job is found, Universal Connector reconnects to that job and satisfies the command line requirements. In this case, that means that if the job has not yet been started, it will be started, Universal Connector will wait for the job to complete (if it has not already), and the output will be returned.

If no matching command ID job is found, Universal Connector will terminate with an error code. Appropriate informational messages will be printed to standard error.

Pre-XBP 2.0 CFT Mode

```
usap -user bob -pwd secret -submit jobdef -start -wait -cmdid 0000000001 -restart yes  
-cft_secure_cft no
```

Secure CFT Mode

```
usap -user bob -pwd secret -submit jobdef -start -wait -cmdid 0000000001 -restart yes  
-cft_secure_cft yes
```

Run a Command ID Job Using Restart AUTO

In the following example, Universal Connector will first scan the SAP system to determine if a matching command ID job exists. If no matching command ID job is found, Universal Connector considers this to be the initial instance of this command ID job and defines the new command ID job to the SAP system. If a matching command ID job is found, Universal Connector reconnects with the existing SAP job.

After determining if the command ID job is initial or a restart, Universal Connector satisfies the command line requirements.

Pre-XBP 2.0 CFT Mode

```
usap -user bob -pwd secret -submit jobdef -start -wait -cft_target_host pwdf0196  
-cft_cmd_prefix "cmd /C" -cmdid 0000000001 -restart auto -cft_secure_cft no
```

Secure CFT Mode

```
usap -user bob -pwd secret -submit jobdef -start -wait -cmdid 0000000001 -restart auto  
-cft_secure_cft yes
```

Working With Pre-defined SAP Jobs

Initial Run of a Command ID Job

The following example will submit, start, and wait for the command ID job defined in the pre-existing SAP job with job name 'JOB_A' and job ID 19561301. Because the RESTART option is set to **no**, Universal Connector will scan the SAP system to ensure that a command ID job with the same job name/command ID pair does not already exist on the system.

If a matching command ID job is found on the SAP system, Universal Connector will exit with an error code before performing the job submission.

Note that the Client Fault Tolerance Command ID Prefix is set up for a Windows host. In many user environments, the Client Fault Tolerance Command ID Prefix parameter can be specified in the configuration file and will never need to be specified on the command line. The same may be true for the Client Fault Tolerance Target Host parameter.

Pre-XBP 2.0 CFT Mode

```
usap -user bob -pwd secret -submit -jobname JOB_A -jobid 19561301 -start -wait  
-cft_target_host pdf0196 -cft_cmd_prefix "cmd /C" -cmdid 0000000001 -restart no  
-cft_secure_cft no
```

Note: The Client Fault Tolerance Command ID Prefix is set up for a Windows host. In many user environments, the Client Fault Tolerance Command ID Prefix parameter can be specified in the configuration file and will never need to be specified on the command line. The same may be true for the Client Fault Tolerance Target Host parameter.

Secure CFT Mode

```
usap -user bob -pwd secret -submit -jobname JOB_A -jobid 19561301 -start -wait  
-cft_secure_cft yes -cft_abap BTCTEST -cmdid 0000000001 -restart no
```

Note: In secure CFT mode, the **cft_secure_cft** and **cft_abap** parameters would most likely be specified in the Universal Connector configuration file.

Restart of a Command ID Job

In the following example, Universal Connector is requested to restart command ID job 0000000001. Universal Connector will scan the SAP system for a matching command ID job.

If a matching command ID job is found, Universal Connector reconnects to that job and satisfies the command line requirements. In this case, that means that if the job has not yet been started, it will be started, Universal Connector will wait for the job to complete (if it hasn't already), and the output will be returned.

If no matching command ID job is found, Universal Connector will terminate with an error code. Appropriate informational messages will be printed to standard error.

Pre-XBP 2.0 CFT Mode

```
usap -user bob -pwd secret -submit -jobname JOB_A -jobid 19561301 -start -wait  
-cmdid 0000000001 -restart yes -cft_secure_cft no
```

Secure CFT Mode

```
usap -user bob -pwd secret -submit -jobname JOB_A -jobid 19561301 -start -wait  
-cmdid 0000000001 -restart yes -cft_secure_cft yes
```

Run a Command ID Job Using Restart AUTO

In the following example, Universal Connector will first scan the SAP system to determine if a matching command ID job exists. If no matching command ID job is found, Universal Connector considers this to be the initial instance of this command ID job and defines the new command ID job to the SAP system. If a matching command ID job is found, Universal Connector reconnects with the existing SAP job.

After determining if the command ID job is initial or a restart, Universal Connector satisfies the command line requirements.

Pre-XBP 2.0 CFT Mode

```
usap -user bob -pwd secret -submit -jobname JOB_A -jobid 19561301 -start -wait  
-cft_target_host pddf0196 -cft_cmd_prefix "cmd /C" -cmdid 0000000001 -restart auto  
-cft_secure_cft no
```

Secure CFT Mode

```
usap -user bob -pwd secret -submit -jobname JOB_A -jobid 19561301 -start -wait  
-cmdid 0000000001 -restart auto -cft_secure_cft yes
```

15.6 Implementing Fault Tolerance Examples

Windows

Implementing Manager Fault Tolerance for Windows

15.6.1 Implementing Manager Fault Tolerance for Windows

Figure 15.4, below, illustrates how to activate manager fault tolerance. A unique command id is always required for manager fault tolerance.

```
ucmd -script script.file -host dallas -encryptedfile encrypted.file
-managerft yes -cmdid uniquejobname -restart auto
```

Figure 15.4 UCMD Manager for Windows - Manager Fault Tolerance

The command is sent to a remote system, **dallas**, for execution. The output of the script is redirected back to the UCMD process. Additional command line options are read from the encrypted file, **encrypted.file**. Manager fault tolerance is turned on. A unique command ID, **uniquejobname**, is coded to identify the process.

Restart is detected automatically. If an executing or pending command ID exists, a reconnect is performed. If not, the process is started as if new.

Command Line Options

The command line options used in this example are:

Options	Description
-script	File from which to read a script file. The script file is sent to the remote system for execution.
-host	Directs the command to a computer with a host name of dallas .
-encryptedfile	File from which to read an encrypted command options file.
-managerft	Specification for whether or not the manager fault tolerant feature is used.
-cmdid	Unique command ID associated the unit of work.
-restart	Specification for whether or not the manager is requesting restart.

Components

Universal Command Manager for Windows

Network Data Transmission

16.1 Overview

Distributed systems, such as Universal Command, communicate over data networks. All Indesca components communicate using the TCP/IP protocol. The UDP protocol is not used for any product data communication over a network.

Indesca 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

Indesca implements the SSL protocol using the OpenSSL library or the IBM z/OS System SSL library, available on the z/OS operating system. The most recent SSL standard is 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 [6.7 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

Indesca components use an application-layer protocol to exchange data messages. The protocol has the following characteristics:

- [Low-Overhead](#)
- [Secure](#)
- [Extensible](#)
- [Configurable Options](#)

The following sections refer to two categories of data transmitted by Indesca:

- Control data (or messages) consists of messages generated by Indesca components in order to communicate with each other. The user of the product has no access to the control data itself.
- 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 Indesca components:

CODE_PAGE

The CODE_PAGE option specifies the code page translation table used to translate network data from and to the local code page for the system on which the program is executing.

A codepage table is text file that contain a two-column table. The table maps local single byte character codes to two-byte UNICODE character codes.

Code pages are located in the product National Language Support (NLS) directory or library. New code pages may be created and added to the NLS directory or library. The CODE_PAGE option value is simply the name of the code page file without any file name extension if present.

CTL_SSL_CIPHER_LIST

The CTL_SSL_CIPHER_LIST option specifies one or more SSL cipher suites that are acceptable to use for network communications on the control session, which is used for component internal communication.

The SSL protocol uses cipher suites to specify the combination of encryption and message digest algorithms used for a session. An ordered list of acceptable cipher suites can be specified in a most to least order of preference.

An example cipher suite list is RC4-MD5,RC4-SHA,AES128-SHA. The RC4-MD5 cipher suite is the most preferred and AES128-SHA is the least preferred.

When 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 `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

Indesca provides network fault tolerance and, in some cases, manager fault tolerance (see [Chapter 15 Fault Tolerance Implementation](#)). These features provide users with the ability to execute jobs that will continue to run when the network is down and when a manager is terminated.

However, there are scenarios in which the user may want to cancel an executing job that supports manager and/or network fault tolerance and have processes terminate immediately. Because of fault tolerance, when the manager is terminated, the server side would begin a connection reestablishment protocol and continue to execute. This would allow the started user job to continue running.

In particular, z/OS supports a CANCEL command that will terminate a job executing on the z/OS operating system.

17.2 z/OS CANCEL Command Support in Universal Command

A user may want to cancel an executing Universal Command job that supports manager and/or network fault tolerance and have both the manager and server processes terminate immediately. Because of fault tolerance, when the manager is terminated, the server side would begin a connection reestablishment protocol and continue to execute. This would allow the started user job to continue running.

When a Universal Command job is cancelled via the z/OS CANCEL command, the job terminates with either of these exit codes:

- Exit code S122, if it is cancelled with a dump.
- Exit code S222, if it is cancelled without a dump.

Part of the responsibility of a Universal Broker executing on a particular host is to monitor the status of all locally running manager processes on that machine. So, when instructed, that Universal Broker could issue a STOP command to the Universal Command Server process associated with the stopped/ended manager process.

17.2.1 Exit Codes

Through the use of the [SERVER_STOP_CONDITIONS](#) configuration option, the Universal Command Manager process notifies the locally running Universal Broker of the exit codes that should cause it to terminate the running Server process. With this option, the user can specify a list of exit codes that should trigger the locally running Universal Broker to issue the STOP command to the manager's Universal Command server-side process.

[SERVER_STOP_CONDITIONS](#) can specify a single exit code or a comma-separated list of exit codes. These stop conditions are passed from the manager to the locally running Universal Broker, which store this and other component-specific data about the executing manager component. When this executing Universal Command Manager process is cancelled or stopped, the locally running Universal Broker detects the ending of the manager process and retrieves its process completion information, which includes the exit code of the manager.

The Universal Broker then compares this exit code with the list of exit codes provided by [SERVER_STOP_CONDITIONS](#). If a match is found, and either network fault tolerance or manager fault tolerance is enabled, the Universal Broker will execute a `uct1` command to STOP the running Universal Command Server component.

17.2.2 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 Command Server process.

This security token is generated on a component-by-component basis by the Universal Broker process that starts the Universal Command Server. Upon generation, this token is returned to the Universal Command Manager which, in turn, updates its locally running Universal Broker with this token. The locally running Universal Broker then uses this token with the issued STOP command to cancel the running Universal Command Server process.

When this token is received by the Universal Broker processes with the request to STOP the server component, the Broker authenticates the received token with the stored token for the running Universal Command Server process. When the token is authenticated, the Universal Command Server process is STOPPED.

17.3 z/OS CANCEL Command Support in Universal Connector

A user may want to cancel an executing Universal Connector job that supports client and / or network fault tolerance and have both the Universal Connector and SAP processes terminate immediately. Because of the separation of work between USAP and SAP, when the Universal Connector client is terminated, the SAP job continues to execute.

In particular, z/OS supports a CANCEL command that will terminate a job executing on the z/OS operating system. When a Universal Connector job is cancelled via the z/OS CANCEL command, the job terminates with either of these exit codes:

- S122, if job is cancelled with a dump.
- S222, if job is cancelled without a dump.

Part of the responsibility of a Universal Broker executing on a particular host is to monitor the status of all locally running manager processes on that machine. So, when instructed, that Universal Broker could invoke a new instance of Universal Connector and issue a CANCEL command to terminate the associated SAP job.

17.3.1 Exit Codes

Through the use of the [SERVER_STOP_CONDITIONS](#) configuration option, the Universal Connector process notifies the locally running Universal Broker of the exit codes that should cause it to terminate the running SAP job. With this option, you can specify a list of exit codes that should trigger the locally running Universal Broker to invoke a Universal Connector process to terminate the SAP job.

[SERVER_STOP_CONDITIONS](#) can specify a single exit code or a comma-separated list of exit codes. These stop conditions are passed from the manager to the locally running Universal Broker, which stores this and other component-specific data about the executing manager component. When this executing Universal Connector process is cancelled or stopped, the locally running Universal Broker detects the ending of the manager process and retrieves its process completion information, which includes the exit code of the manager.

The Universal Broker then compares this exit code with the list of exit codes provided by [SERVER_STOP_CONDITIONS](#). If a match is found, the Universal Broker will invoke a new instance of the Universal Connector to execute a CANCEL command to terminate the running SAP job.

Glossary

This glossary defines terms used within the Indesca business solution:

Agent

A single Indesca (or Infitran) installation comprised of one Universal Broker and one or more Stonebranch Solutions components, such as Universal Command.)

API

API (Application Programming Interface) is a set of functions, procedures, methods, classes, or protocols that an operating system, library, or service provides to support requests made by computer programs.

Asynchronous Communication

Transmission of data or sending of messages without the need to wait for a reply from the destination before continuing with the next operation.

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**liley **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

Indesca component that provides client services initiating requests on behalf of the user (for example, a Universal Command manager batch job requesting the execution of a command on a remote server).

Message

Abstract format, or container, for sending data between applications or services. No implementation is implied.

Message-Based Application

A message-based application accesses a target application by sending a message to a queue that is controlled by the target application. This queue must be known and accessible to the application sending the message.

Message-Based Workload

Any application, service, or function that supports a message-based communication protocol such as JMS or MQ.

Message Exchange Pattern

A Message Exchange Pattern (MEP) describes the pattern of messages required by a communications protocol in order to establish or use a communication channel.

Message-based application

Application accessed via a web browser over the Internet or Intranet.

MQ Connector

MQ connector supports workload execution via the MQ messaging protocol using synchronous and asynchronous communication.

PKI

PKI (**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

Indesca component initiated either by a client or the Universal Broker. All servers are started by the Universal Broker. A manager can request that the Broker initiates a server on its behalf, and the manager and server then work together to perform a service, or a server can be started automatically by the Broker when the Broker starts and stopped when the Broker stops.

SOA

SOA (**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 Indesca components.

E-MAIL

All Locations

support@stonebranch.com

Customer support contact via e-mail also can be made via the Stonebranch website:

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

C

- CA certificate *306*
- CA-Driver procedure
 - UCMD Manager for z/OS *80*
- certificate
 - CA (Certificate Authority) *306*
 - creating *307*
- checking logs *173*
- command coded as a script
 - UCMD Manager for z/OS *343*
- configuration
 - remote *374*
- configuration files
 - reading *391*
 - Universal Broker *393*
- configuring
 - logging operations *173*
- console application *417*
- creating
 - CA certificate *306*
 - certificate *307*

D

- database directory *484*
- databases
 - Universal Event Monitor *484*

E

- error messages
 - translating into return codes *452*
- event log (Windows)

- writing records *476*
- execute a Windows .bat file
 - UCMD Manager for z/OS *75, 76*
- executing
 - Universal Broker for Windows *417*

F

- files
 - input to Universal Message Translator *452*

H

- HP NonStop
 - configuration refresh *404*
 - console application *422*
 - daemon *423*

I

- IBM i
 - enabling Universal Broker *421*
 - ending Universal Broker *421*
 - removing *ALLOBJ authority *248*
 - starting Universal Broker *421*
- I-Management Console
 - remote configuration *374*

L

- logging
 - configuring operations *173*
 - levels *173*
- logs *173*

M

maintaining

- component server data 483

managed mode 375

- selecting 375

manager fault tolerance

- UCMD Manager for z/OS 78

O

override command line parameters from execute statement

- UCMD Manager for z/OS 74

override standard files with DDNAME

- UCMD Manager for z/OS 72

override standard files with SYMBOLICS

- UCMD Manager for z/OS 73

Q

querying Universal Broker 469

R

reading

- configuration files 391

redirect standard out to file and UCMD Manager

- UCMD Manager for z/OS 69, 70

remote configuration 374

- I-Management Console 374

- managed mode 375

- unmanaged mode 374

S

selecting

- managed mode 375

start-up modes 377

T

translating error messages into return codes 452

U

unique command ID with OPC

- UCMD Manager for z/OS 84

unique command ID with Zeke

- UCMD Manager for z/OS 83

Universal Broker

- start-up 377

Universal Event Monitor

- databases 484

Universal Message Translator

- input files 452

Universal Spool 483

UNIX

- configuration refresh 400

- console application 419

- daemon process 418

- databases 493

- unmanaged mode 374

W

Windows

- databases 494

- environment 417

- Universal Access Control List 246

- writing event log records 476

Windows service 417

- writing event log records (Windows) 476

X

- X.509 certificates 277

Z

z/OS

- databases 493

- starting Universal Broker 416

- stopping Universal Broker 416



**950 North Point Parkway, Suite 200
Alpharetta, Georgia 30005
U.S.A.**

