Stonebranch Solutions

Version 4.2.0

Universal Data Mover

Reference Guide

udm-ref-4201

stonebranch

Universal Data Mover

Reference Guide

Stonebranch Solutions 4.2.0

Document Name	Universal Data Mover 4.2.0 Reference Guide				
Document ID	**udm-ref-4201**				
Components	**z/OS**	**UNIX**	**Windows**	**IBM i**	**HP NonStop**
Universal Data Mover Manager	√	√	√	√	
Universal Data Mover Server	√	√	√	√	

Stonebranch Documentation Policy

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

Summary of Changes

Changes for Universal Data Mover 4.2.0 Reference Guide (udm-ref-4201) October 29, 2010

- Removed requirement for licensed version of Universal Command in Sections 14.18 exec and 19.2 exec Command

Changes for Universal Data Mover 4.2.0 Reference Guide (udm-ref-4200) August 6, 2010

Universal Data Mover 4.2.0.0

- Moved detailed technical information from Universal Data Mover 4.1.0 User Guide into Universal Data Mover 4.2.0 Reference Guide.

  Information on component features and examples was moved to the Infitran 4.2.0 User Guide.
- Added Section 6.31 ENCRYPT in 6 Universal Data Mover Manager Configuration Options.
- Added Configuration Options table entries and Command Line Syntax entries for ENCRYPT in the following chapters:
  - 2 Universal Data Mover Manager for z/OS
  - 3 Universal Data Mover Manager for Windows
  - 4 Universal Data Mover Manager for UNIX
  - 5 Universal Data Mover Manager for IBM i

Changes for Universal Data Mover 4.1.0 Reference Guide (udm-ref-4100) February 10, 2010

- Added _execrc built-in variable in Table 21.7 Built-In Variables.
- Specified, in 14 UDM Commands, that the openlog and savedata commands do not support the HFS file system.

Universal Data Mover 4.1.0.0

- Specified the TCP_NO_DELAY configuration option for OS/400 in 6 Universal Data Mover Manager Configuration Options.
- Specified the TCP_NO_DELAY configuration option for OS/40 in 11 Universal Data Mover Server Configuration Options.

Changes for Universal Data Mover 3.2.0 Reference Guide (udm-ref-3206) September 8, 2009

Universal Data Mover 3.2.0.6

- Specified information about added support for the UTF-8 codepage in:
  - UDM Manager CODE_PAGE configuration option
  - UDM Server CODE_PAGE configuration option
- Added the UDM Server LOGON_METHOD configuration option.
- Added the following code pages in Section 21.12 Character Code Pages:
  - IBM875
  - IBM4971

Changes for Universal Data Mover 3.2.0 Reference Guide (udm-ref-3205) July 29, 2009

Universal Data Mover 3.2.0.1 for OS/400

- Modified document for upgrade from Universal Data Mover 3.1.1 for OS/400 to Universal Data Mover 3.2.0 for OS/400, including:
  - Changed the following OS/400 names throughout the document:
    - Universal Broker subsystem name from **`UBROKER`** to **`UNVUBR320`**.
    - Universal Broker user profile name from **`UBROKER`** to **`UNVUBR320`**.
    - Universal Products installation library name from **`UNIVERSAL`** to **`UNVPRD320`**.
    - Universal Products spool library name from **`UNVSPOOL`** to **`UNVSPL320`**.
    - Universal Products temporary directory from **`UNVTMP`** to **`UNVTMP320`**.

- Added the following OS/400 configuration option in 6 Universal Data Mover Manager Configuration Options:
  - CODEPAGE_TO_CCSID_MAP
- Specified the following configuration options for OS/400 in 6 Universal Data Mover Manager Configuration Options:
  - ACTIVITY_MONITORING
  - CA_CERTIFICATES
  - CERTIFICATE
  - CERTIFICATE_REVOCATION_LIST
  - COMMENT
  - EVENT_GENERATION
  - OPEN_RETRY
  - OPEN_RETRY_COUNT
  - OPEN_RETRY_INTERVAL
  - PLF_DIRECTORY
  - PRIVATE_KEY
  - PRIVATE_KEY_PWD
  - PROXY_CERTIFICATES
- Added a STRUDM parameter to the following configuration options for OS/400 in 6 Universal Data Mover Manager Configuration Options:
  - CTL_SSL_CIPHER_LIST
  - DATA_SSL_CIPHER_LIST
  - FRAME_INTERVAL
  - MODE_TYPE
  - OUTBOUND_IP
- Added character translation information for OS/400 to the following configuration option in 6 Universal Data Mover Manager Configuration Options:
  - PRIVATE_KEY_PWD
- Added the following OS/400 configuration option in 11 Universal Data Mover Server Configuration Options.
  - CODEPAGE_TO_CCSID_MAP
- Specified the following configuration options for OS/400 in 11 Universal Data Mover Server Configuration Options:
  - ACTIVITY_MONITORING
  - EVENT_GENERATION
  - TMP_DIRECTORY
- Added the following Universal Access Control List entry for OS/400 in 13 Universal Data Mover UACL Entries:
  - UDM_MGR_ACCESS

Changes for Universal Data Mover 3.2.0 Reference Guide (udm-ref-3204) April 1, 2009

- Added an example of the upper command.

Universal Data Mover 3.2.0.3

- Added the TCP_NO_DELAY configuration option in 6 Universal Data Mover Manager Configuration Options.
- Added the TCP_NO_DELAY configuration option in 11 Universal Data Mover Server Configuration Options.
- Added the following commands in 14 UDM Commands:
  - appenddata
  - closelog
  - echolog
  - logdata
  - move
  - openlog
  - savedata
- Added the following parameters to the exec command in 14 UDM Commands:
  - stdout
  - stderr
- Added the following attributes in Table 14.4 Common File System Attributes:
  - srccreatetime
  - srcmodtime
  - srcaccesstime
- Added the following variables in Section 21.6 Built-In Variables:
  - _uuid
  - _lastmsg
- Added Section 21.7 _file Built-in Variable – Special Attributes.

Changes for Universal Data Mover 3.2.0 Reference Guide (udm-ref-3203) December 17, 2008

- Specified, in the following tables, that the `trans` attribute of the attrib command is valid only under the `hfs` file system for z/OS and OS/400:
  - Table 14.4 Common File System Attributes
  - Table 21.2 Common File System Attributes
- Changed the name of the environment variable for the Universal Data Mover Manager SYSTEM_ID configuration option from **`UDMSYSTEM`** to **`UDMSYSTEMID`**.

Changes for Universal Data Mover 3.2.0 Reference Guide (udm-ref-3202) October 17, 2008

- Added a note about incorrect character translations for the Universal Data Mover Manager for OS/400 PRIVATE_KEY_PWD option.
- Changed **JCL SNTYPE** value to **type** for the **dsntype** attribute in Table 21.3 z/OS attrib Command - Dynamic Allocation Attributes.

Changes for Universal Data Mover 3.2.0 Reference Guide (udm-ref-3201) September 5, 2008

- Added toll-free telephone number for North America in A Customer Support.

Changes for Universal Data Mover 3.2.0 Reference Guide (udm-ref-320) May 16, 2008

Universal Data Mover 3.2.0.3

- Consolidated UDM Manager options and UDM Manager Invocation options into a single chapter, 6 Universal Data Mover Manager Configuration Options.
- Added the following configuration options in 6 Universal Data Mover Manager Configuration Options:
  - ACTIVITY_MONITORING
  - BIF_DIRECTORY
  - CA_CERTIFICATES
  - CERTIFICATE
  - CERTIFICATE_REVOCATION_LIST
  - COMMENT
  - EVENT_GENERATION
  - OPEN_RETRY_COUNT
  - OPEN_RETRY_INTERVAL
  - PLF_DIRECTORY
  - PRIVATE_KEY
  - PRIVATE_KEY_PWD
  - PROXY_CERTIFICATES
  - SAF_KEY_RING
  - SAF_KEY_RING_LABEL
  - SERVER_STOP_CONDITIONS
  - SSL_IMPLEMENTATION

    - SYSTEM_ID
- Added the following configuration options in 11 Universal Data Mover Server Configuration Options.
    - ACTIVITY_MONITORING
    - EVENT_GENERATION
    - TMP_DIRECTORY
- Added 12 Universal Data Mover Component Definition Options.
- Added 13 Universal Data Mover UACL Entries.
- Added 17 new commands in 14 UDM Commands.
- Modified the following commands in 14 UDM Commands:
    - attrib: Added the **`mode`** attribute to list of Common File System Attributes.
    - open: Added **`comment`** parameter.
    - set.
- Removed the following specification methods for all UDM Server configuration options:
    - Command Line, Short Form
    - Command Line, Long Form
    - Environment Variable
- Added Configuration File Keyword as a specification method for Windows options.

Contents

List of Figures

List of Tables

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

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

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\help.txt`**).

Command Line Syntax Diagrams

Command line syntax diagrams use the following conventions:

Convention	Description
`bold monospace font`	Specifies values to be typed verbatim, such as file / data set names.
`italic monospace font`	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.

Tips from the Stoneman

Look to the Stoneman for suggestions
or for any other information
that requires special attention.

Stoneman's Tip

Vendor References

References are made throughout 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 are used within this document:

- **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.

Note: These names do not imply software support in any manner. For a detailed list of supported operating systems, see the Stonebranch Solutions 4.2.0 Installation Guide.

Document Organization

- Overview (Chapter 1)
  General architectural and functional overview of Universal Data Mover.
- Universal Data Mover Manager for z/OS (Chapter 2)
  Description of Universal Data Mover Manager specific to the z/OS operating system.
- Universal Data Mover Manager for Windows (Chapter 3)
  Description of Universal Data Mover Manager specific to the Windows operating system.
- Universal Data Mover Manager for UNIX (Chapter 4)
  Description of Universal Data Mover Manager specific to the UNIX operating system.
- Universal Data Mover Manager for IBM i (Chapter 5)
  Description of Universal Data Mover Manager specific to the IBM i operating system.
- Universal Data Mover Manager Configuration Options (Chapter 6)
  Detailed information about the configuration options used with the Universal Data Mover Manager.
- Universal Data Mover Server for z/OS (Chapter 7)
  Description of Universal Data Mover Server specific to the z/OS operating system.
- Universal Data Mover Server for Windows (Chapter 8)
  Description of Universal Data Mover Server specific to the Windows operating system.
- Universal Data Mover Server for UNIX (Chapter 9)
  Description of Universal Data Mover Server specific to the UNIX operating system.
- Universal Data Mover Server for IBM i (Chapter 10)
  Description of Universal Data Mover Server specific to the IBM i operating system.
- Universal Data Mover Server Configuration Options (Chapter 11)
  Detailed information about the configuration options used with the Universal Data Mover Server.
- Universal Data Mover Component Definition Options (Chapter 12)
  Detailed information about the component definition options used with the Universal Data Mover.
- Universal Data Mover UACL Entries (Chapter 13)
  Detailed information about the Universal Access Control List (UACL) entries available for use with the Universal Data Mover.
- UDM Commands (Chapter 14)
  Detailed information about the Universal Data Mover commands available for use with Universal Data Mover.
- UDM Scripting Language (Chapter 15)
  Description of the Universal Data Mover scripting language.
- UDM Transfer Operations (Chapter 16)
  General description of the Universal Data Mover transfer operations.
- Transfer Operations (z/OS-Specific) (Chapter 17)
  Description of the Universal Data Mover transfer operations specific to the z/OS operating system.

- Transfer Operations (IBM i-Specific) (Chapter 18)
  General description of the Universal Data Mover transfer operations specific to the IBM i operating system.
- Remote Execution (Chapter 19)
  Description of remote execution procedures for Universal Data Mover.
- Return Code Processing (Chapter 20)
  Description of Universal Data Mover return code processing.
- Additional Information (Chapter 21)
  Additional technical information relative to Universal Data Mover.
- Customer Support (Appendix A)
  Customer support contact information for users of Universal Data Mover.

CHAPTER 1

Overview

1.1 Introduction to Universal Data Mover

Universal Data Mover is Infitran's managed file transfer component that manages data in a secure and automated manner, allowing data to be transferred between any platforms in your environment and initiated from any platform.

Every Universal Data Mover transfer operation is comprised of three components: manager, primary server, and secondary server.

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

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

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

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

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

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

1.2 Transfer Components

There are three components to any UDM transfer operation:

1. Manager
2. Primary server
3. Secondary server

The manager may act as the primary server, depending on the type of transfer session: two-party or three-party (see Section 1.3 Transfer Sessions). The secondary server is always a separate and distinct component invoked via the Universal Broker.

1.2.1 Manager

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

1.2.2 Primary Server

When a transfer session is being established, the UDM Manager invokes the primary server, which acts as the first endpoint in a transfer operation. In turn, the primary server invokes the secondary server, providing a single path of communication. The primary server also acts as a relay for the UDM Manager, forwarding on any messages for the secondary server from the UDM Manager. This single message pipeline reduces the number of connections needed for three-party transfers (see Section 1.3.3 Three-Party Transfer Sessions).

1.2.3 Secondary Server

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

1.3 Transfer Sessions

As discussed in Section 1.2 Transfer Components, transfer operations take place within the context of a transfer session. A transfer operation is initiated once the UDM Manager has established a transfer session with the primary and secondary transfer servers. All subsequent transfer operations take place between the primary and secondary transfer servers.

UDM transfer sessions can be either two-party or three-party.

1.3.1 Logical Names

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

1.3.2 Two-Party Transfer Sessions

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

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

(See Figure 1.1 UDM Transfer Sessions.)

1.3.3 Three-Party Transfer Sessions

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

(See Figure 1.1 UDM Transfer Sessions.)

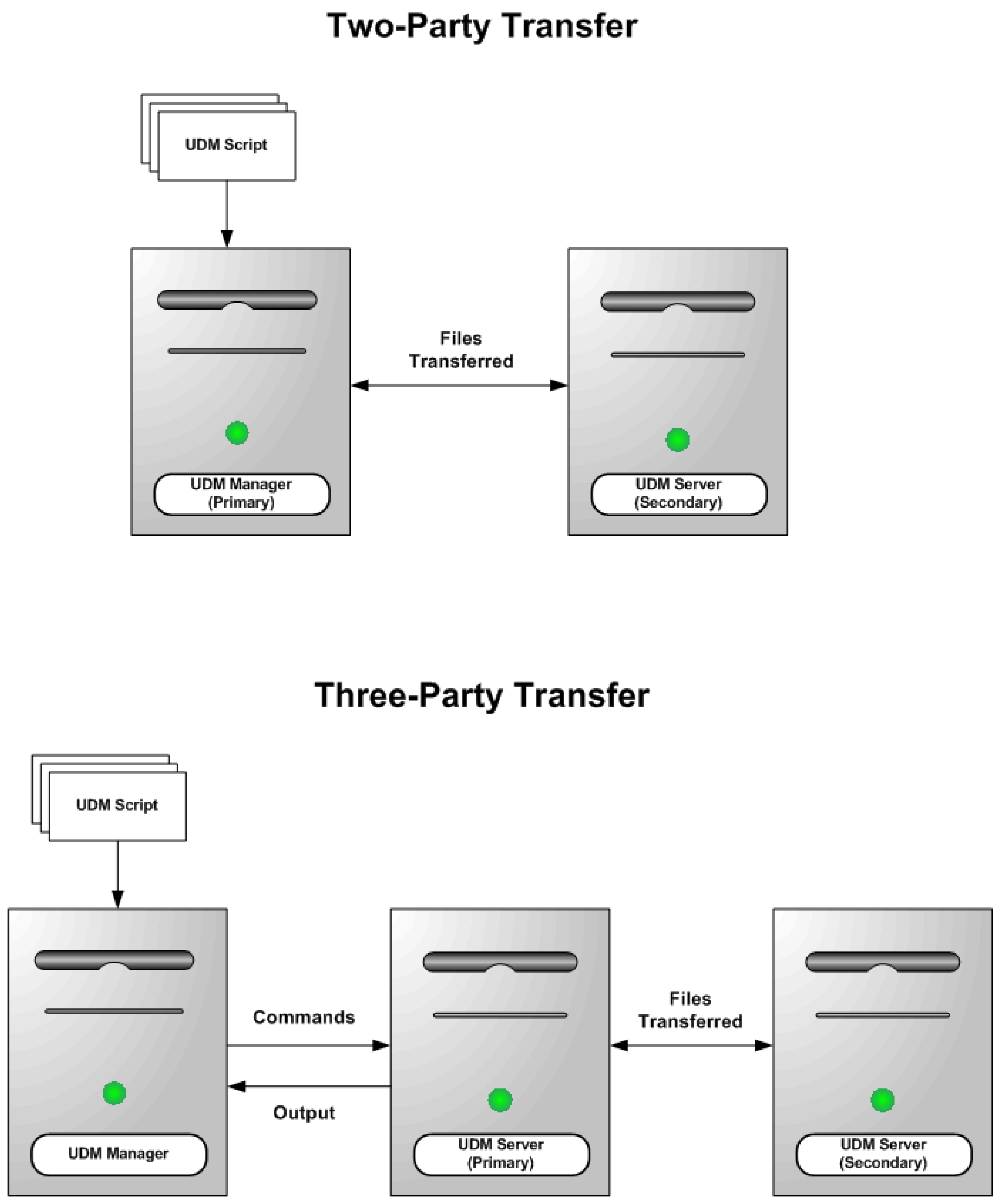


Figure 1.1 UDM Transfer Sessions

CHAPTER 2

Universal Data Mover Manager for z/OS

2.1 Overview

This chapter provides information on the Universal Data Mover (UDM) Manager, specific to the z/OS operating system.

UDM Manager transfers files between any computers running UDM Server. Using a UDM command script, you indicate to the UDM Manager the actions to take. The UDM Manager connects to the UDM Server (or Servers) and processes your request.

The z/OS Batch Manager provides a batch job interface to remote computers running the UDM Server component.

2.2 Usage

UDM Manager for z/OS executes as a batch job.

This section describes the JCL, configuration and configuration options, and command line syntax of UDM Manager for z/OS.

2.2.1 JCL Procedure

Figure 2.1, below, illustrates the Universal Data Mover for z/OS JCL procedure (**UDMPRC**, located in the **SUNVSAMP** library) that is provided to simplify the execution JCL and future maintenance.

```
//UDMPRC   PROC UPARM=,                -- UDM options
//             USPRFC=USPRFC00,        -- USAP SAP RFC member
//             UNVPRE=#SHLQ.UNV,
//             UNVPRD=#PHLQ.UNV
//*
//PS1      EXEC PGM=UDM,REGION=256M,PARM='ENVAR(TZ=EST5EDT)/&UPARM'
//STEPLIB  DD  DISP=SHR,DSN=&UNVPRE..SUNVLOAD
//*
//UNVNLS   DD  DISP=SHR,DSN=&UNVPRE..SUNVNLS
//UNVUSRC  DD  DISP=SHR,DSN=&UNVPRD..UNVCONF(&USPRFC)
//UNVCLIB  DD  DISP=SHR,DSN=&UNVPRE..SUNVSAMP
//*
//UNVTRACE DD  SYSOUT=*
//UNVTRMDL DD  DISP=SHR,DSN=&UNVPRD..MDL
//*
//SYSPRINT DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//*
//SYSIN    DD  DUMMY                   -- UDM command options
//UNVSCR   DD  DUMMY                   -- UDM script
```

Figure 2.1 UDM Manager for z/OS – JCL Procedure

For this JCL procedure:

- UPARM parameter is used to specify EXEC PARM keyword values.
- UNVPRE parameter specifies the data set name prefix of Stonebranch Solutions installation data sets.
- UNVPRD parameter specifies the data set name prefix of Stonebranch Solutions production data sets.

2.2.2 DD Statements in JCL

Table 2.1, below, describes the DD statements used in the Universal Data Mover for z/OS JCL illustrated in Figure 2.1.

ddname	DCB Attributes	Mode	Description
STEPLIB	DSORG=PO, RECFM=U	input	Stonebranch Solutions load library containing the program being executed.
UNVNLS	DSORG=PO, RECFM=(F, FB, V, VB)	input	Stonebranch Solutions national language support library. Contains message catalogs and code page translation tables.
UNVUSRC	DSORG=PS, RECFM=(F, FB, V, VB)	input	Universal SAP connector RFC member.
UNVCLIB	DSORG=PO, RECFM=(F, FB, V, VB)	input	UDM call library: UDM searches for script files specified on the call command in this library.
UNVTRACE	DSORG=PS, RECFM=(F, FB, V, VB)	Output	UDM trace output.
UNVTRMDL	DSORG=PS, RECFM=(F, FB, V, FB)	input	Data set used as a model for creating UCMD and USAP trace files when they are called by UDM using the exec and execsap commands.
SYSPRINT	DSORG=PS, RECFM=(F, FB, V, VB)	output	Standard output file for the UDM program.
SYSOUT	DSORG=PS, RECFM=(F, FB, V, VB)	output	Standard error file for the UDM program.
SYSIN	DSORG=PS, RECFM=(F, FB, V, VB)	input	Standard input file for the UDM program. UDM reads its command options from SYSIN.
UNVSCR	DSORG=PS, RECFM=(F, FB, V, VB)	input	UDM command script: UDM executes the script allocated to this ddname.
The C runtime library determines the default DCB attributes. Refer to the IBM manual *OS/390 C/C++ Programming Guide* for details on default DCB attributes for stream I/O.			

Table 2.1 UDM Manager for z/OS – DD Statements in JCL

2.2.3 JCL

Figure 2.2, below, illustrates the Universal Data Mover for z/OS JCL using the **UDMPRC** procedure illustrated in Figure 2.1.

```
//jobname  JOB CLASS=A,MSGCLASS=X
//STEP1    EXEC UDMPRC
//UNVSCR   DD  *
 open srv=sol7 user=id001 pwd=pwd001
 copy local='uid.data' srv=data
 quit
/*
```

Figure 2.2 UDM Manager for z/OS – JCL

Job step STEP1 executes the procedure **UDMPRC**.

The UDM script commands are specified on the **UNVSCR** DD.

2.2.4 Configuration

Configuration consists of:

- Setting default options and preferences for all executions of UDM Manager.
- Setting options and preferences for a single execution of UDM Manager.

Configuration options are read from the following sources:

1. PARM keyword
2. SYSIN ddname
3. Configuration file

The order of precedence is the same as the list above; PARM keyword options being the highest and configuration file being the lowest. That is, options specified via a PARM keyword override options specified via a SYSIN ddname, and so on.

Detailed information on these methods of configuration can be found in Chapter 8 Configuration Management of the Infitran 4.2.0 User Guide.

Configuration File

The configuration file provides the simplest method of specifying configuration options whose values will not change with each command invocation. These default values are used if the options are not read from one or more other sources.

Some options only can be specified in the configuration file; they have no corresponding command line equivalent. Other options cannot be specified in the configuration file; they must be specified via one or more other sources for a single execution of UDM Manager.

2.2.5 Configuration Options

Table 2.2, below, describes the configuration options used to execute UDM Manager for z/OS.

Each **Option Name** is a link to detailed information about that configuration option.

Option Name	Description
ACTIVITY_MONITORING	Specification for whether or not product activity monitoring events are generated.
ALLOC_ABNORMAL_DISP	Abnormal disposition of a data set being allocated.
ALLOC_BLKSIZE	Block size used for newly allocated data sets.
ALLOC_DATACLAS	SMS data class used for newly allocated data sets.
ALLOC_DIR_BLOCKS	Number of directory blocks for newly allocated partitioned data sets.
ALLOC_DSORG	Data set organization used for newly allocated data sets.
ALLOC_INPUT_STATUS	Status of data sets being allocated for input.
ALLOC_LRECL	Logical record length used for newly allocated data sets.
ALLOC_MGMTCLAS	SMS management class used for newly allocated data sets.
ALLOC_NORMAL_DISP	Normal disposition of a data set being allocated.
ALLOC_OUTPUT_STATUS	Status of existing data sets being allocated for output.
ALLOC_PRIM_SPACE	Primary space allocation used for newly allocated data sets.
ALLOC_RECFM	Record format used for newly allocated data sets.
ALLOC_SEC_SPACE	Secondary space allocation used for newly allocated data sets.
ALLOC_SPACE_UNIT	Space unit in which space is allocated for newly allocated data sets.
ALLOC_STORCLAS	SMS storage class used for newly allocated data sets.
ALLOC_UNIT	Unit used for newly allocated data sets.
ALLOC_VOLSER	Volume serial number used for newly allocated data sets.
CA_CERTIFICATES	File name / ddname of the PEM-formatted trusted CA X.509 certificates.
CERTIFICATE	File name / ddname of UDM Manager’s PEM-formatted X.509 certificate.
CERTIFICATE_REVOCATION_LIST	File name / ddname of the PEM-formatted CRL.
CODE_PAGE	Character code page used to translate text data received and transmitted over the network.
COMMENT	User-defined string.
CTL_SSL_CIPHER_LIST	Acceptable and preferred SSL cipher suites to use for the control session between UDM components.
DATA_COMPRESSION	Specification for whether or not data is compressed on all standard I/O files.

Option Name	Description
ENCRYPT	Encryption method to be used in a UDM transfer session if one is not specified by an encrypt parameter in the UDM open command.
DATA_SSL_CIPHER_LIST	Acceptable and preferred SSL cipher suites to use for the data session on which file data is transferred between UDM primary and secondary servers.
EVENT_GENERATION	Events to be generated as persistent events.
FRAME_INTERVAL	Number of UDM transfer blocks transferred before a frame-sync message is sent when UDM is operating with network fault tolerance turned on.
HELP	Writes a description of the command options and their format.
IDLE_TIMEOUT	Number of seconds of inactivity in an interactive UDM session after which the manager will close the session.
KEEP_ALIVE_INTERVAL	Default interval at which a keep alive message is sent from the manager to the transfer server(s).
MERGE_LOG	Specification for whether or not to merge standard out and standard error output streams from a remote command to the UDM transaction log.
MESSAGE_LANGUAGE	Universal Message Catalog (UMC) file used to write messages.
MESSAGE_LEVEL	Level of messages to write.
MODE_TYPE	Default transfer mode type for UDM sessions.
NETWORK_DELAY	Expected network latency.
NETWORK_FAULT_TOLERANT	Specification for whether or not UDM transfer sessions are network fault tolerant by default.
OPEN_RETRY	Level of fault tolerance for the open command.
OPEN_RETRY_COUNT	Maximum number of attempts that will be made to establish a session by the open command.
OPEN_RETRY_INTERVAL	Number of seconds that UDM will wait between each open retry attempt.
OUTBOUND_IP	Host or IP address that UDM binds to when initiating outgoing connections to another UDM server.
PRIVATE_KEY	ddname of Manager’s PEM formatted RSA private key.
PRIVATE_KEY_PWD	Password for the Manager’s PRIVATE_KEY.
PROXY_CERTIFICATES	Specification for whether or not UDM will use proxy certificates in three-party transfer sessions if a certificate is supplied to the UDM Manager.
RECONNECT_RETRY_COUNT	Number of attempts the manager will make to re-establish a transfer session when a network fault occurs.
RECONNECT_RETRY_INTERVAL	Number of seconds that UDM will wait between each successive attempt to re-establish a transfer session when a network fault occurs.
RECV_BUFFER_SIZE	Size of the TCP receive buffer for UDM.
REMOTE_PORT	TCP port number on the remote computer used for invoking UDM Server instances.
SAF_KEY_RING	SAF certificate key ring name.

Option Name	Description
SAF_KEY_RING_LABEL	SAF key ring certificate label.
SCRIPT	ddname from which to read a UDM script command file.
SCRIPT_OPTIONS	Options to pass to the script command file.
SEND_BUFFER_SIZE	Size of the TCP send buffer for UDM.
SERVER_STOP_CONDITIONS	Exit codes that cause Universal Broker to cancel the corresponding UDM Server of the exited UDM Manager.
SSL_IMPLEMENTATION	SSL implementation.
SYSTEM_ID	Local Universal Broker with which the UDM Manager must register.
TCP_NO_DELAY	Specification for whether or not to use TCP packet coalescing.
TRACE_FILE_LINES	Maximum number of lines to write to the trace file.
TRACE_TABLE	Size of a wrap-around trace table maintained in memory.
UMASK	File mode creation mask.
VERSION	Writes the program version information and copyright.

Table 2.2 UDM Manager for z/OS - Configuration Options

2.2.6 Command Line Syntax

Figure 2.3 and Figure 2.4, below, illustrate the command line syntax of UDM Manager for z/OS.

```
udm
[-alloc_abnormal_disp {keep|delete|catlg|uncatlg}]
[-alloc_blksize size]
[-alloc_dataclas class]
[-alloc_dir_blocks number]
[-alloc_dsorg {po|ps}]
[-alloc_input_status {old|shr}]
[-alloc_lrecl length]
[-alloc_mgmtclas class]
[-alloc_normal_disp {keep|delete|catlg|uncatlg}]
[-alloc_output_status {old|shr}]
[-alloc_prim_space space]
[-alloc_recfm format]
[-alloc_sec_space space]
[-alloc_space_unit {cyl|trk|number}]
[-alloc_storclas class]
[-alloc_unit unit]
[-alloc_volser number]
[-system_id ID]
[-ssl_implementation {openssl|system}]
[-ca_certs ddname]
[-cert ddname]
[-private_key ddname [-private_key_pwd password] ]
[-proxy_certificates {yes|no}]
[-crl ddname]
[-script ddname]
[-options options]
[-codepage codepage]
[-ctl_ssl_cipher_list list]
[-data_ssl_cipher_list list]
[-encrypt({yes|no|cipher})]
[-compress {yes|no}[,{zlib|hasp}] ]
[-delay seconds]
[-idle_timeout seconds]
[-keep_alive_interval seconds]
[-lang language]
[-level {trace|audit|info|warn|error}[,{time|notime}] ]
```

Figure 2.3 UDM Manager for z/OS - Command Line Syntax (1 of 2)

```
[-network_fault_tolerant {yes|no} [-frame_interval number] ]
[-mode_type {binary|text}]
[-umask number]
[-outboundip host]
[-port port]
[-recvbuffersize size]
[-open_retry {yes|no}]
[-open_retry_count number]
[-open_retry_interval number]
[-retry_count number]
[-retry_interval seconds]
[-sendbuffersize size]
[-saf_key_ring name]
[-saf_key_ring_label label]
[-server_stop_conditions codes]
[-tcp_no_delay option]
[-tracefilelines number]
[-trace_table size,{error|always|never}]
[-comment text]

udm
{-help | -version}
```

Figure 2.4 UDM Manager for z/OS - Command Line Syntax (2 of 2)

CHAPTER 3

Universal Data Mover Manager for Windows

3.1 Overview

This chapter provides information on the Universal Data Mover (UDM) Manager, specific to the Windows operating system.

UDM Manager transfers files between any computers running UDM Server. Using a UDM command script, you indicate to the UDM Manager the actions to take. The UDM Manager connects to the UDM Server (or Servers) and processes your request.

3.2 Usage

The UDM Manager command is executed from the command line or a script. After it has been initiated, UDM is ready to process commands. The commands can come from standard input or a script file.

This section describes the modes of operation, configuration and configuration options, and command line syntax of UDM Manager for Windows.

3.2.1 Modes of Operation

Under Windows, UDM can be run either in:

- Interactive mode
- Batch mode

Running UDM in Interactive Mode

To invoke UDM in interactive mode, enter the following at the command prompt:

```
udm
```

This will start the UDM Manager. You will be greeted with a startup message and the UDM prompt, similar to this:

```
UNV2800I Universal Data Mover 4.2.0 Level 0 started.
udm>
```

At the **udm>** prompt, you can enter any UDM command.

To exit UDM, enter the following command at the **udm>** prompt:

```
quit
```

Running UDM in Batch Mode

When running in batch mode, UDM processes a script file.

When the script file has finished executing, UDM will exit automatically:

```
udm -s script_filename
```

3.2.2 Configuration

Configuration consists of:

- Setting default options and preferences for all executions of UDM Manager.
- Setting options and preferences for a single execution of UDM Manager.

Configuration options are read from the following sources:

1. Command line
2. Command file
3. Environment variables
4. Configuration file

The order of precedence is the same as the list above; command line options being the highest and configuration file being the lowest. That is, options specified via the command line override options specified via a command file, and so on.

Detailed information on these methods of configuration can be found in Chapter 8 Configuration Management of the Infitran 4.2.0 User Guide.

Configuration File

The configuration file provides the simplest method of specifying configuration options whose values will not change with each command invocation. These default values are used if the options are not read from one or more other sources.

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.

Some options only can be specified in the configuration file; they have no corresponding command line equivalent. Other options cannot be specified in the configuration file; they must be specified via one or more other sources for a single execution of UDM Manager.

3.2.3 Configuration Options

Table 3.1, below, describes the configuration options used to execute UDM Manager for Windows.

Each **Option Name** is a link to detailed information about that configuration option.

Option Name	Description
ACTIVITY_MONITORING	Specification for whether or not product activity monitoring events are generated.
CA_CERTIFICATES	File name / ddname of the PEM-formatted trusted CA X.509 certificates.
CERTIFICATE	File name / ddname of UDM Manager's PEM-formatted X.509 certificate.
CERTIFICATE_REVOCATION_LIST	File name / ddname of the PEM-formatted CRL.
CODE_PAGE	Character code page used to translate text data received and transmitted over the network.
COMMENT	User-defined string.
CTL_SSL_CIPHER_LIST	Acceptable and preferred SSL cipher suites to use for the control session between UDM components.
DATA_COMPRESSION	Specification for whether or not data is compressed on all standard I/O files.
ENCRYPT	Encryption method to be used in a UDM transfer session if one is not specified by an encrypt parameter in the UDM open command.
DATA_SSL_CIPHER_LIST	Acceptable and preferred SSL cipher suites to use for the data session on which file data is transferred between UDM primary and secondary servers.
EVENT_GENERATION	Events to be generated as persistent events.
FRAME_INTERVAL	Number of UDM transfer blocks transferred before a frame-sync message is sent when UDM is operating with network fault tolerance turned on.
HELP	Writes a description of the command options and their format.
IDLE_TIMEOUT	Number of seconds of inactivity in an interactive UDM session after which the manager will close the session.
INSTALLATION_DIRECTORY	Directory on which UDM Manager is installed.
KEEP_ALIVE_INTERVAL	Default interval at which a keep alive message is sent from the manager to the transfer server(s).
MERGE_LOG	Specification for whether or not to merge standard out and standard error output streams from a remote command to the UDM transaction log.
MESSAGE_LANGUAGE	Universal Message Catalog (UMC) file used to write messages.
MESSAGE_LEVEL	Level of messages to write.
MODE_TYPE	Default transfer mode type for UDM sessions.
NETWORK_DELAY	Expected network latency.

Option Name	Description
NETWORK_FAULT_TOLERANT	Specification for whether or not UDM transfer sessions are network fault tolerant by default.
NLS_DIRECTORY	Directory where the UDM Manager message catalog and code page tables are located.
OPEN_RETRY	Level of fault tolerance for the open command.
OPEN_RETRY_COUNT	Maximum number of attempts that will be made to establish a session by the open command.
OPEN_RETRY_INTERVAL	Number of seconds that UDM will wait between each open retry attempt.
OUTBOUND_IP	Host or IP address that UDM binds to when initiating outgoing connections to another UDM server.
PRIVATE_KEY	ddname of Manager's PEM formatted RSA private key.
PRIVATE_KEY_PWD	Password for the Manager's PRIVATE_KEY.
PROXY_CERTIFICATES	Specification for whether or not UDM will use proxy certificates in three-party transfer sessions if a certificate is supplied to the UDM Manager.
RECONNECT_RETRY_COUNT	Number of attempts the manager will make to re-establish a transfer session when a network fault occurs.
RECONNECT_RETRY_INTERVAL	Number of seconds that UDM will wait between each successive attempt to re-establish a transfer session when a network fault occurs.
RECV_BUFFER_SIZE	Size of the TCP receive buffer for UDM.
REMOTE_PORT	TCP port number on the remote computer used for invoking UDM Server instances.
SCRIPT_FILE	Script file containing UDM commands to execute.
SCRIPT_OPTIONS	Options to pass to the script command file.
SEND_BUFFER_SIZE	Size of the TCP send buffer for UDM.
TCP_NO_DELAY	Specification for whether or not to use TCP packet coalescing.
TRACE_FILE_LINES	Maximum number of lines to write to the trace file.
TRACE_TABLE	Size of a wrap-around trace table maintained in memory.
UCMD_PATH	Sets the complete path to UCMD for calls by the exec command.
UMASK	File mode creation mask.
VERSION	Writes the program version information and copyright.

Table 3.1 UDM Manager for Windows - Configuration Options

3.2.4 Command Line Syntax

Figure 3.1, below, illustrates the command line syntax of UDM Manager for Windows.

```
udm
[-ca_certs file]
[-cert file]
[-private_key file [-private_key_pwd password] ]
[-proxy_certificates {yes|no}]
[-crl file]
[-script filename]
[-options options]
[-codepage codepage]
[-ctl_ssl_cipher_list list]
[-data_ssl_cipher_list list]
[-encrypt({yes|no|cipher})]
[-compress {yes|no}[,{zlib|hasp}] ]
[-delay seconds]
[-idle_timeout seconds]
[-keep_alive_interval seconds]
[-lang language]
[-level {trace|audit|info|warn|error}[,{time|notime}] ]
[-network_fault_tolerant {yes|no} [-frame_interval number] ]
[-mode_type {binary|text}]
[-umask number]
[-outboundip host]
[-port port]
[-recvbuffersize size]
[-open_retry {yes|no}]
[-open_retry_count number]
[-open_retry_interval number]
[-retry_count number]
[-retry_interval seconds]
[-sendbuffersize size]
[-tcp_no_delay option]
[-tracefilelines number]
[-trace_table size,{error|always|never}]
[-comment text]

udm
{-help | -version}
```

Figure 3.1 UDM Manager for Windows - Command Line Syntax

CHAPTER 4

Universal Data Mover Manager for UNIX

4.1 Overview

This chapter provides information on the Universal Data Mover (UDM) Manager, specific to the UNIX operating system.

UDM Manager transfers files between any computers running UDM Server. Using a UDM command script, you indicate to the UDM Manager the actions to take. The UDM Manager connects to the UDM Server (or Servers) and processes your request.

4.2 Usage

The UDM Manager command is executed from the command line or a script. After it has been initiated, UDM is ready to process commands. The commands can come from standard input or a script file.

This section describes the modes of operation, configuration and configuration options, and command line syntax of UDM Manager for UNIX.

4.2.1 Modes of Operation

Under UNIX, UDM can be run either in:

- Interactive mode
- Batch mode

Running UDM in Interactive Mode

To invoke UDM in interactive mode, enter the following at the command prompt:

```
udm
```

This will start the UDM Manager. You will be greeted with a start-up message and the UDM prompt, similar to this:

```
UNV2800I Universal Data Mover 4.2.0 Level 0 started.
udm>
```

At the **udm>** prompt, you can enter ny UDM command.

To exit UDM, enter the following command at the **udm>** prompt:

```
quit
```

Running UDM in Batch Mode

UDM also can be run in batch mode. When running in batch mode, UDM processes a script file. When the script file has finished executing, UDM will exit automatically:

```
udm -s script_filename
```

4.2.2 Configuration

Configuration consists of:

- Setting default options and preferences for all executions of UDM Manager.
- Setting options and preferences for a single execution of UDM Manager.

Configuration options are read from the following sources:

1. Command line
2. Command file
3. Environment variables
4. Configuration file

The order of precedence is the same as the list above; command line options being the highest and configuration file being the lowest. That is, options specified via the command line override options specified via a command file, and so on.

Detailed information on these methods of configuration can be found in Chapter 8 Configuration Management of the Infitran 4.2.0 User Guide.

Configuration File

The configuration file, **`udm.conf`**, provides the simplest method of specifying configuration options whose values will not change with each command invocation. These default values are used if the options are not read from one or more other sources. (See the Stonebranch Solutions 4.2.0 Installation Guide to determine the directory in which it is located.)

Some options only can be specified in the configuration file; they have no corresponding command line equivalent. Other options cannot be specified in the configuration file; they must be specified via one or more other sources for a single execution of UDM Manager.

4.2.3 Configuration Options

Table 4.1, below, describes the configuration options used to execute UDM Manager for UNIX.

Each **Option Name** is a link to detailed information about that configuration option.

Option Name	Description
ACTIVITY_MONITORING	Specification for whether or not product activity monitoring events are generated.
BIF_DIRECTORY	Broker Interface File directory where the Universal Broker interface file is located.
CA_CERTIFICATES	File name / ddname of the PEM-formatted trusted CA X.509 certificates.
CERTIFICATE	File name / ddname of UDM Manager's PEM-formatted X.509 certificate.
CERTIFICATE_REVOCATION_LIST	File name / ddname of the PEM-formatted CRL.
CODE_PAGE	Character code page used to translate text data received and transmitted over the network.
COMMENT	User-defined string.
CTL_SSL_CIPHER_LIST	Acceptable and preferred SSL cipher suites to use for the control session between UDM components.
DATA_COMPRESSION	Specification for whether or not data is compressed on all standard I/O files.
ENCRYPT	Encryption method to be used in a UDM transfer session if one is not specified by an encrypt parameter in the UDM open command.
DATA_SSL_CIPHER_LIST	Acceptable and preferred SSL cipher suites to use for the data session on which file data is transferred between UDM primary and secondary servers.
EVENT_GENERATION	Events to be generated as persistent events.
FRAME_INTERVAL	Number of UDM transfer blocks transferred before a frame-sync message is sent when UDM is operating with network fault tolerance turned on.
HELP	Writes a description of the command options and their format.
IDLE_TIMEOUT	Number of seconds of inactivity in an interactive UDM session after which the manager will close the session.
INSTALLATION_DIRECTORY	Directory in which Universal Data Mover is installed.
KEEP_ALIVE_INTERVAL	Default interval at which a keep alive message is sent from the manager to the transfer server(s).
MERGE_LOG	Specification for whether or not to merge standard out and standard error output streams from a remote command to the UDM transaction log.
MESSAGE_LANGUAGE	Universal Message Catalog (UMC) file used to write messages.
MESSAGE_LEVEL	Level of messages to write.
MODE_TYPE	Default transfer mode type for UDM sessions.

Option Name	Description
NETWORK_DELAY	Expected network latency.
NETWORK_FAULT_TOLERANT	Specification for whether or not UDM transfer sessions are network fault tolerant by default.
NLS_DIRECTORY	Directory where the UDM Manager message catalog and code page tables are located.
OPEN_RETRY	Level of fault tolerance for the open command.
OPEN_RETRY_COUNT	Maximum number of attempts that will be made to establish a session by the open command.
OPEN_RETRY_INTERVAL	Number of seconds that UDM will wait between each open retry attempt.
OUTBOUND_IP	Host or IP address that UDM binds to when initiating outgoing connections to another UDM server.
PLF_DIRECTORY	Program Lock File directory that specifies the location of the UDM Manager program lock file.
PRIVATE_KEY	ddname of Manager’s PEM formatted RSA private key.
PRIVATE_KEY_PWD	Password for the Manager’s PRIVATE_KEY.
PROXY_CERTIFICATES	Specification for whether or not UDM will use proxy certificates in three-party transfer sessions if a certificate is supplied to the UDM Manager.
RECONNECT_RETRY_COUNT	Number of attempts the manager will make to re-establish a transfer session when a network fault occurs.
RECONNECT_RETRY_INTERVAL	Number of seconds that UDM will wait between each successive attempt to re-establish a transfer session when a network fault occurs.
RECV_BUFFER_SIZE	Size of the TCP receive buffer for UDM.
REMOTE_PORT	TCP port number on the remote computer used for invoking UDM Server instances.
SCRIPT_FILE	Script file containing UDM commands to execute.
SCRIPT_OPTIONS	Parameters to pass to the script file.
SEND_BUFFER_SIZE	Size (in bytes) of the TCP send buffer for UDM.
TCP_NO_DELAY	Specification for whether or not to use TCP packet coalescing.
TRACE_FILE_LINES	Maximum number of lines to write to the trace file.
TRACE_TABLE	Size of a wrap-around trace table maintained in memory.
UCMD_PATH	Sets the complete path to UCMD for calls by the exec command.
UMASK	File mode creation mask.
USAP_PATH	Sets the complete path to USAP for calls by the execsap command.
VERSION	Writes the program version information and copyright.

Table 4.1 UDM Manager for UNIX - Configuration Options

4.2.4 Command Line Syntax

Figure 4.1, below, illustrates the command line syntax of UDM Manager for UNIX.

```
udm
[-bif_directory directory]
[-plf_directory directory]
[-ca_certs file]
[-cert file]
[-private_key file [-private_key_pwd password] ]
[-proxy_certificates {yes|no}]
[-crl file]
[-script filename]
[-options options]
[-codepage codepage]
[-ctl_ssl_cipher_list list]
[-data_ssl_cipher_list list]
[-encrypt({yes|no|cipher})]
[-compress {yes|no}[,{zlib|hasp}] ]
[-delay seconds]
[-idle_timeout seconds]
[-keep_alive_interval seconds]
[-lang language]
[-level {trace|audit|info|warn|error}[,{time|notime}] ]
[-network_fault_tolerant {yes|no} [-frame_interval number] ]
[-mode_type {binary|text}]
[-umask number]
[-outboundip host]
[-port port]
[-recvbuffersize size]
[-open_retry {yes|no}]
[-open_retry_count number]
[-open_retry_interval number]
[-retry_count number]
[-retry_interval seconds]
[-sendbuffersize size]
[-tcp_no_delay option]
[-tracefilelines number]
[-trace_table size,{error|always|never}]
[-comment text]

udm
{-help | -version}
```

Figure 4.1 UDM Manager for UNIX - Command Line Syntax

CHAPTER 5

Universal Data Mover Manager for IBM i

5.1 Overview

This chapter provides information on the Universal Data Mover (UDM) Manager, specific to the IBM i operating system.

UDM Manager transfers files between any computers running UDM Server. Using a UDM command script, you indicate to the UDM Manager the actions to take. The UDM Manager connects to the UDM Server (or Servers) and processes your request.

5.2 Usage

The UDM Manager command is invoked from a command line or a batch job. After it has been initiated, UDM is ready to process commands.

The commands can come from:

- Standard input or a script file in interactive mode
- Script file in batch mode

This section describes the modes of operation, configuration and configuration options, and command line syntax of UDM Manager for UNIX.

5.2.1 Stonebranch Solutions for IBM i Commands

The names of the Stonebranch Solutions for IBM i commands that are installed in the IBM i **QSYS** library are tagged with the Stonebranch Solutions for IBM i **v**ersion / **r**elease / **m**odification number, **420**. The names of the commands installed in the Stonebranch Solutions for IBM i product library, **UNVPRD420**, are untagged.

To maintain consistency across releases, you may prefer to use the untagged names in your production environment. The **UCHGRLS** (Change Release Tag) program lets you change the tagged command names in **QSYS** to the untagged command names in **UNVPRD420**.

(See the Stonebranch Solutions 4.2.0 Installation Guide for detailed information on **UCHGRLS**.)

This chapter references the IBM i commands by their untagged names. If you are using commands with tagged names to run UDM, substitute the tagged names for the untagged names in these references.

5.2.2 Modes of Operation

Under IBM i, UDM can be run either:

- In interactive mode
- From a script file
- As a batch application

Additionally under IBM i, UDM can use either the LIB or HFS file system.

Running UDM Interactively

To invoke UDM as an interactive application:

1. Enter the following on the command line: **`STRUDM`**
2. Press <Enter>.

This will start the UDM Manager. You will be greeted with a startup message and the **`udm>`** prompt, similar to this:

```
UNV2800I Universal Data Mover 4.2.0 Level 0 started.

udm>
```

At the **`udm>`** prompt, you can enter any UDM command.

To exit UDM, enter the following command at the **`udm>`** prompt:

```
quit
```

Running UDM from a Script

To invoke a UDM script:

1. Enter the following on the command line:
   `STRUDM SCRFILE(library/file) SCRMBR(member)`
2. Press <Enter>.

This will start the UDM Manager using the script located by library, file, and member. When the script file has finished executing, UDM will exit automatically.

UDM requires the member name; there is no default. Requiring the member name makes script specification under IBM i behave as it does on other platforms. On other systems, there is no default search order as exists under IBM i. However, users may explicitly provide **`*FILE`** as a member name to use the IBM i default file search order.

Running UDM in Batch Mode

UDM also can be run in batch mode. When running in batch mode, use a script as shown in Section Running UDM from a Script.

To execute a batch file such as the one below, use:

SBMDBJOB FILE(LIBNAME/FILENAME) MBR(MBRNAME):

```
 //BCHJOB JOB(MYUDMJOB) ENDSEV(10)
STRUDM MYLIB/QSCRSRC UDM817
 //ENDBCHJOB
```

Output is sent to the output queue associated with the batch job. Two spooled files will be sent to the output queue; one file associated with standard out and one file associated with standard error.

5.2.3 Configuration

Configuration consists of:

- Setting default options and preferences for all executions of UDM Manager.
- Setting options and preferences for a single execution of UDM Manager.

Configuration options are read from the following sources:

1. Command line
2. Command file
3. Environment variables
4. Configuration file

The order of precedence is the same as the list above; command line options being the highest and configuration file being the lowest. That is, options specified via the command line override options specified via a command file, and so on.

Detailed information on these methods of configuration can be found in Chapter 8 Configuration Management of the Infitran 4.2.0 User Guide.

Configuration File

The configuration file provides the simplest method of specifying configuration options whose values will not change with each command invocation. These default values are used if the options are not read from one or more other sources.

The installation default for the UDM configuration file is Stonebranch Solutions installation library **UNVPRD420,** file **UNVCONF**, and member **UDM**. The configuration file name can be any valid file name. It can be edited manually using an IBM i editor such as SEU, EDTF, or any other installed source file editor, or via the IFS using a text editor. If a text editor is used to edit the file via the IFS, the padded spaced must be removed for lines that exceed the file maximum record length.

Some options only can be specified in the configuration file; they have no corresponding command line equivalent. Other options cannot be specified in the configuration file; they must be specified via one or more other sources for a single execution of UDM Manager.

5.2.4 Configuration Options

Table 5.1, below, describes the configuration options used to execute UDM Manager for IBM i.

Each **Option Name** is a link to detailed information about that configuration option.

Option Name	Description
ACTIVITY_MONITORING	Specification for whether or not product activity monitoring events are generated.
CA_CERTIFICATES	File name of the PEM-formatted trusted CA X.509 certificates.
CERTIFICATE	File name of UDM Manager's PEM-formatted X.509 certificate.
CERTIFICATE_REVOCATION_LIST	File name of the PEM-formatted CRL.
CODE_PAGE	Character code page used to translate text data received and transmitted over the network.
CODEPAGE_TO_CCSID_MAP	Specification to use either the internal or external table for code page to CCSID mapping.
COMMENT	User-defined string.
CTL_SSL_CIPHER_LIST	Acceptable and preferred SSL cipher suites to use for the control session between UDM components.
DATA_COMPRESSION	Specification for whether or not data is compressed on all standard I/O files.
ENCRYPT	Encryption method to be used in a UDM transfer session if one is not specified by an encrypt parameter in the UDM open command.
DATA_SSL_CIPHER_LIST	Acceptable and preferred SSL cipher suites to use for the data session on which file data is transferred between UDM primary and secondary servers.
EVENT_GENERATION	Events to be generated as persistent events.
FRAME_INTERVAL	Number of UDM transfer blocks transferred before a frame-sync message is sent when UDM is operating with network fault tolerance turned on.
IDLE_TIMEOUT	Number of seconds of inactivity in an interactive UDM session after which the manager will close the session.
KEEP_ALIVE_INTERVAL	Default interval at which a keep alive message is sent from the manager to the transfer server(s).
MERGE_LOG	Specification for whether or not to merge standard out and standard error output streams from a remote command to the UDM transaction log.
MESSAGE_LANGUAGE	Universal Message Catalog (UMC) file used to write messages.
MESSAGE_LEVEL	Level of messages to write.
MODE_TYPE	Default transfer mode type for UDM sessions.
NETWORK_DELAY	Expected network latency.
NETWORK_FAULT_TOLERANT	Specification for whether or not UDM transfer sessions are network fault tolerant by default.

Option Name	Description
OPEN_RETRY	Level of fault tolerance for the open command.
OPEN_RETRY_COUNT	Maximum number of attempts that will be made to establish a session by the open command.
OPEN_RETRY_INTERVAL	Number of seconds that UDM will wait between each open retry attempt.
OUTBOUND_IP	Host or IP address that UDM binds to when initiating outgoing connections to another UDM server.
PLF_DIRECTORY	Program Lock File directory that specifies the location of the UDM Manager program lock file.
PRIVATE_KEY	Location of Manager's PEM formatted RSA private key.
PRIVATE_KEY_PWD	Password for the Manager's PRIVATE_KEY.
PROXY_CERTIFICATES	Specification for whether or not UDM will use proxy certificates in three-party sessions if a certificate is supplied to the UDM Manager.
RECONNECT_RETRY_COUNT	Number of attempts the manager will make to re-establish a transfer session when a network fault occurs.
RECONNECT_RETRY_INTERVAL	Number of seconds that UDM will wait between each successive attempt to re-establish a transfer session when a network fault occurs.
RECV_BUFFER_SIZE	Size of the TCP receive buffer for UDM.
REMOTE_PORT	TCP port number on the remote computer used for invoking UDM Server instances.
SCRIPT_FILE	Script file containing UDM commands to execute.
SCRIPT_OPTIONS	Parameters to pass to the script file.
SEND_BUFFER_SIZE	Size of the TCP send buffer for UDM.
SIZE_ATTRIB	Default file creation size for physical files of both data and source file types.
TCP_NO_DELAY	Specification for whether or not to use TCP packet coalescing.
TRACE_FILE_LINES	Maximum number of lines to write to the trace file.
TRACE_TABLE	Size of a wrap-around trace table maintained in memory.
UMASK	File mode creation mask.
VERSION	Writes the program version information and copyright.

Table 5.1 UDM Manager for IBM i - Configuration Options

5.2.5 Command Line Syntax

Figure 5.1, below, illustrates the command line syntax of UDM Manager for IBM i.

```
STRUDM
[SCRFILE([library/]filename) [SCRMBR(member) ]
[PLFDIR (directory)]
[OPTIONS(options)]
[CODEPAGE(codepage)]
[CTLCPHRLST(cipherlist)]
[DTACPHRLST(cipherlist)]
[ENCRYPT({*yes|*no|cipher})]
[COMPRESS(*{yes|no})]
[CMPRSMTH(*{zlib|hasp})]
[DELAY(seconds)]
[IDLTIMOUT(seconds)]
[KEEPALIVE(seconds)]
[MODETYPE (*{bin|binary|text}]
[MSGLANG(language)]
[MSGLEVEL(*{trace|audit|info|warn|error}) [,*{yes|no}] ]
[NETWORKFT(*{yes|no}) [FRAMEINT(number)]]
[OPENRETRY(*{yes|no} count interval)]
[OUTBOUNDIP(host)]
[PORT(port)]
[PROXYCERT(option)]
[RCVBUFSIZE(size)]
[RETRYCNT(number)]
[RETRYINT(seconds)]
[SNDBUFSIZE(size)]
[TCPNODELAY(*{yes|no})]
[TRCLINES(number)]
[TRCTBL(size,*{error|always|never})]
[CACERTS(file [lib] ) [CACERTSMBR(member)] ]
[CERT(file [lib] ) [CERTMBR(member)]
     PVTKEYF(file [lib] ) [PVTKEYFMBR(member)] [PVTKEYPWD(password)] ]
[CRLFILE(file [lib]) [CRLMBR(member)] ]
[COMMENT(user-defined string)]

STRUDM
[VERSION(*{yes|no})]
```

Figure 5.1 UDM Manager for IBM i - Command Line Syntax

CHAPTER 6

Universal Data Mover Manager Configuration Options

6.1 Overview

This chapter provides detailed information on the configuration options available for use with the Universal Data Mover Manager.

The options are listed alphabetically, without regard to any specific operating system.

Section 6.2 Configuration Options Information provides a guideline for understanding the information presented or each option.

6.2 Configuration Options Information

For each configuration option, this chapter provides the following information.

Description

Describes the configuration option and how it is used.

Usage

Provides a table of the following information:

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	<Format / Value>					
Command Line, Long Form	<Format / Value>					
Environment Variable	<Format / Value>					
Configuration File Keyword	<Format / Value>					
STRUDM Parameter	<Format / Value>					

Method

Identifies the different methods used to specify Universal Data Mover Manager configuration options:

- Command Line Option, Short Form
- Command Line Option, Long Form
- Environment Variable
- Configuration File Keyword
- STRUDM Parameter

Note: Each option can be specified using one or more methods.

Syntax

Identifies the syntax of each method that can be used to specify the option:

• Format	Specific characters that identify the option.
• Value	Type of value(s) to be supplied for this method.

Note: If a Method is not valid for specifying the option, the Syntax field contains **n/a**.

(Operating System)

Identifies (with a ✓) the operating systems for which each method of specifying the option is valid:

- IBM i
- NonStop (HP NonStop)
- UNIX
- Windows
- z/OS

Values

Identifies all possible values for the specified value type.

Defaults are identified in **[bracketed bold type]**.

<Additional Information>

Identifies any additional information specific to the option.

6.3 Configuration Options List

Table 6.1, below, identifies all UDM Manager configuration options.

Option	Description	Page
ACTIVITY_MONITORING	Specification for whether or not product activity monitoring events are generated.	78
ALLOC_ABNORMAL_DISP	Disposition of data set when an abnormal ending occurs.	79
ALLOC_BLKSIZE	Block size used for newly allocated data sets.	80
ALLOC_DATACLAS	SMS data class used for newly allocated data sets.	81
ALLOC_DIR_BLOCKS	Number of directory blocks for newly allocated partitioned data sets.	82
ALLOC_DSORG	Data set organization used for newly allocated data sets.	83
ALLOC_INPUT_STATUS	Status of data sets being allocated for input.	84
ALLOC_LRECL	Logical record length used for newly allocated data sets.	85
ALLOC_MGMTCLAS	SMS management class used for newly allocated data sets.	86
ALLOC_NORMAL_DISP	Disposition of data set when normal ending occurs.	87
ALLOC_OUTPUT_STATUS	Status of existing data sets being allocated for output.	88
ALLOC_PRIM_SPACE	Primary space allocation used for newly allocated data sets.	89
ALLOC_RECFM	Record format used for newly allocated data sets.	90
ALLOC_SEC_SPACE	Secondary space allocation used for newly allocated data sets.	91
ALLOC_SPACE_UNIT	Space unit in which space is allocated for newly allocated data sets.	92
ALLOC_STORCLAS	SMS storage class used for newly allocated data sets.	93
ALLOC_UNIT	Unit used for newly allocated data sets.	94
ALLOC_VOLSER	Volume serial number used for newly allocated data sets.	95
BIF_DIRECTORY	Broker Interface Directory that specifies the location of the Universal Broker interface file.	96
CA_CERTIFICATES	File name / ddname of the PEM-formatted trusted CA X.509 certificates.	97
CERTIFICATE	File name / ddname of UDM Manager’s PEM-formatted X.509 certificate.	98
CERTIFICATE_REVOCATION_LIST	File name / ddname of the PEM-formatted CRL.	99
CODE_PAGE	Character code page used to translate text data.	100
CODEPAGE_TO_CCSID_MAP	Specification to use the internal or external table for code page to CCSID mapping.	102
COMMENT	User-defined string.	104
CTL_SSL_CIPHER_LIST	Acceptable and preferred SSL cipher suites to use for the control session between UDM components.	105

Option	Description	Page
DATA_COMPRESSION	Specification for whether or not data is compressed on all standard I/O files.	106
DATA_SSL_CIPHER_LIST	SSL cipher suites to use for data session between UDM primary and secondary servers.	109
ENCRYPT	Encryption method to be used in a UDM transfer session if one is not specified by an encrypt parameter in the UDM open command.	108
EVENT_GENERATION	Events to be generated as persistent events.	110
FRAME_INTERVAL	Number of UDM transfer blocks transferred before a frame-sync message is sent with network fault tolerance turned on.	112
HELP	Prints a description of the command options and their format.	113
IDLE_TIMEOUT	Number of seconds of inactivity in an interactive UDM session after which the UDM Manager will close the session.	114
INSTALLATION_DIRECTORY	Directory in which UDM Manager is installed.	115
KEEP_ALIVE_INTERVAL	Default interval at which a keep alive message is sent from the UDM Manager to the transfer server(s).	116
MERGE_LOG	Specification for merging standard out and standard error output streams from a remote command to the UDM transaction log.	117
MESSAGE_LANGUAGE	Universal Message Catalog (UMC) file used to format messages.	118
MESSAGE_LEVEL	Level of messages that UDM will write.	119
MODE_TYPE	Default transfer mode type for UDM sessions.	121
NETWORK_DELAY	Expected network latency (in seconds).	122
NETWORK_FAULT_TOLERANT	Setting for whether or not UDM transfer sessions are network fault tolerant by default.	123
NLS_DIRECTORY	Directory name where the UDM Manager can find its message catalog and code page tables.	124
OPEN_RETRY	Level of fault tolerance for the open command.	125
OPEN_RETRY _COUNT	Maximum number of attempts that will be made to establish a session by the open command.	127
OPEN_RETRY_INTERVAL	Number of seconds that UDM will wait between each open retry attempt.	128
OUTBOUND_IP	Host or IP address that UDM binds to when initiating outgoing connections to another UDM server.	129
PLF_DIRECTORY	Program Lock File directory that specifies the location of the UDM Manager program lock file.	130
PRIVATE_KEY	ddname of Manager's PEM formatted RSA private key.	131
PRIVATE_KEY_PWD	Password for the Manager's PRIVATE_KEY.	132
PROXY_CERTIFICATES	Specification for whether or not UDM will use proxy certificates in three-party transfer sessions if a certificate is supplied to the UDM Manager.	132

Option	Description	Page
RECONNECT_RETRY_COUNT	Number of attempts that the UDM Manager will make to re-establish a transfer session when a network fault occurs.	134
RECONNECT_RETRY_INTERVAL	Number of seconds that UDM will wait between each successive attempt to re-establish a transfer session when a network fault occurs.	135
RECV_BUFFER_SIZE	Size of the TCP receive buffer for UDM.	136
REMOTE_PORT	TCP port number on the remote computer used for invoking UDM Server instances.	137
SAF_KEY_RING	SAF certificate key ring name.	138
SAF_KEY_RING_LABEL	SAF key ring certificate label.	139
SCRIPT	ddname from which to read a UDM script command file.	140
SCRIPT_FILE	Script file containing UDM commands to execute.	141
SCRIPT_OPTIONS	Script options to pass into the script command file.	142
SEND_BUFFER_SIZE	Size of the TCP send buffer for UDM.	143
SERVER_STOP_CONDITIONS	Exit codes that cause Universal Broker to cancel the corresponding UDM Server of the exited UDM Manager.	144
SIZE_ATTRIB	Default size for file creation of physical files for both data and source file types.	145
SSL_IMPLEMENTATION	SSL implementation.	146
SYSTEM_ID	Local Universal Broker with which the UDM Manager must register.	147
TCP_NO_DELAY	Specification for whether or not to use TCP packet coalescing.	148
TRACE_FILE_LINES	Maximum number of lines to write to the trace file.	149
TRACE_TABLE	Size of a wrap-around trace table maintained in memory.	151
UCMD_PATH	Complete path to Universal Command for calls by the **exec** command.	153
UMASK	File mode creation mask.	154
USAP_PATH	Complete path to USAP for calls by the **execsap** command.	155
VERSION	Specification for writing of program version information and copyright.	156

Table 6.1 UDM Manager Configuration Options

6.4 ACTIVITY_MONITORING

Description

The ACTIVITY_MONITORING option specifies whether or not product activity monitoring events are generated.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	n/a					
Environment Variable	n/a					
Configuration File Keyword	activity_monitoring *option*	√		√	√	√
STRUDM Parameter	n/a					

Values

option is the specification for whether or not product activity monitoring events are generated.

Valid values for *option* are:

- **yes**
  Activate product activity monitoring events
- **no**
  Deactivate product activity monitoring events

[Default is yes.]

6.5 ALLOC_ABNORMAL_DISP

Description

The ALLOC_ABNORMAL_DISP option is a dynamic allocation option that specifies the disposition of data set when an abnormal ending occurs.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-alloc_abnormal_disp *disposition*					√
Environment Variable	n/a					
Configuration File Keyword	alloc_abnormal_disp *disposition*					√
STRUDM Parameter	n/a					

Values

disposition is equivalent to the third positional parameter of the JCL DD statement's DISP parameter.

Valid values for *disposition* are:

- **keep**
  Keep the data set.
- **delete**
  Delete the data set.
- **catlg**
  Catalog the data set.
- **uncatlg**
  Un-catalog the data set.

[Default is delete.]

References

Refer to the IBM JCL Reference manual for complete details.

6.6 ALLOC_BLKSIZE

Description

The ALLOC_BLKSIZE option is a dynamic allocation option that specifies the block size used for newly allocated data sets.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-alloc_blksize *size*					√
Environment Variable	n/a					
Configuration File Keyword	alloc_blksize *size*					√
STRUDM Parameter	n/a					

Values

size is equivalent to the JCL DD statement's BLKSIZE parameter.

Valid values for *size* are any number (size of a block).

[Default is *27998*.]

References

Refer to the IBM JCL Reference manual for complete details.

6.7 ALLOC_DATACLAS

Description

The ALLOC_BLKSIZE option is a dynamic allocation option that specifies the SMS data class used for newly allocated data sets.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-alloc_dataclas *class*					√
Environment Variable	n/a					
Configuration File Keyword	alloc_dataclas *class*					√
STRUDM Parameter	n/a					

Values

class is equivalent to the JCL DD statement's DATACLAS parameter.

Valid values for *class* are any SMS data classes defined in the local environment.

[There is no default.]

References

Refer to the IBM JCL Reference manual for complete details.

6.8 ALLOC_DIR_BLOCKS

Description

The ALLOC_DIR_BLOCKS option is a dynamic allocation option that specifies the number of directory blocks for newly allocated partitioned data sets.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-alloc_dir_blocks *number*					√
Environment Variable	n/a					
Configuration File Keyword	alloc_dir_blocks *number*					√
STRUDM Parameter	n/a					

Values

number is equivalent to the third positional parameter of the second positional parameter of the JCL DD statement's SPACE parameter.

Valid values for *number* are any number (number of directory blocks to allocate).

[Default is *20*.]

References

Refer to the IBM JCL Reference manual for complete details.

6.9 ALLOC_DSORG

Description

The ALLOC_DSORG option is a dynamic allocation option that specifies the data set organization used for newly allocated data sets.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-alloc_dsorg *organization*					√
Environment Variable	n/a					
Configuration File Keyword	alloc_dsorg *organization*					√
STRUDM Parameter	n/a					

Values

organization is equivalent to the JCL DD statement's DSORG parameter.

Valid values for *organization* are:

- **po**
  Partitioned organization
- **ps**
  Physically sequential

[Default is ps.]

References

Refer to the IBM JCL Reference manual for complete details.

6.10 ALLOC_INPUT_STATUS

Description

The ALLOC_INPUT_STATUS option is a dynamic allocation option that specifies the status of data sets being allocated for input.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-alloc_input_status *status*					√
Environment Variable	n/a					
Configuration File Keyword	alloc_input_status *status*					√
STRUDM Parameter	n/a					

Values

status is equivalent to the first positional parameter of the JCL DD statement's DISP parameter.

Valid values for *status* are:

- **old**
  Allocate the data set exclusively.
- **shr**
  Allocate the data set non-exclusively.

[Default is old.]

References

Refer to the IBM JCL Reference manual for complete details.

6.11 ALLOC_LRECL

Description

The ALLOC_LRECL option is a dynamic allocation option that specifies the logical record length used for newly allocated data sets.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-alloc_lrecl *length*					√
Environment Variable	n/a					
Configuration File Keyword	alloc_lrecl *length*					√
STRUDM Parameter	n/a					

Values

length is equivalent to the first positional parameter of the JCL DD statement's LRECL parameter.

Valid values for *length* are any number (length of the record).

[Default is *1024*.]

References

Refer to the IBM JCL Reference manual for complete details.

6.12 ALLOC_MGMTCLAS

Description

The ALLOC_MGMTCLAS option is a dynamic allocation option that specifies the SMS management class used for newly allocated data sets.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-alloc_mgmtclas *class*					√
Environment Variable	n/a					
Configuration File Keyword	alloc_mgmtclas *class*					√
STRUDM Parameter	n/a					

Values

class is equivalent to the first positional parameter of the JCL DD statement's MGMTCLAS parameter.

Valid values for *class* are any SMS management classes defined in the local environment.

[There is no default.]

References

Refer to the IBM JCL Reference manual for complete details.

6.13 ALLOC_NORMAL_DISP

Description

The ALLOC_NORMAL_DISP option is a dynamic allocation option that specifies the disposition of data set when normal ending occurs.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-alloc_normal_disp *disposition*					√
Environment Variable	n/a					
Configuration File Keyword	alloc_normal_disp *disposition*					√
STRUDM Parameter	n/a					

Values

disposition is equivalent to the second positional parameter of the JCL DD statement's DISP parameter.

Valid values for *disposition* are:

- **keep**
  Keep the data set.
- **delete**
  Delete the data set.
- **catlg**
  Catalog the data set.
- **uncatlg**
  Un-catalog the data set.

[Default is catlg.]

References

Refer to the IBM JCL Reference manual for complete details.

6.14 ALLOC_OUTPUT_STATUS

Description

The ALLOC_OUTPUT_STATUS option is a dynamic allocation option that specifies the status of data sets being allocated for output.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-alloc_output_status *status*					√
Environment Variable	n/a					
Configuration File Keyword	alloc_output_status *status*					√
STRUDM Parameter	n/a					

Values

status is equivalent to the first positional parameter of the JCL DD statement's DISP parameter.

Valid values for *status* are:

- **new**
  Create new data set.
- **shr**
  Allocate the data set non-exclusively.
- **old**
  Allocate the data set exclusively.
- **mod**
  Either create a new data set, for exclusive use, or allocate a sequential data set exclusively and add records to the end of it.

[Default is old.]

References

Refer to the IBM JCL Reference manual for complete details.

6.15 ALLOC_PRIM_SPACE

Description

The ALLOC_PRIM_SPACE option is a dynamic allocation option that specifies the primary space allocation used for newly allocated data sets.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-alloc_prim_space *space*					√
Environment Variable	n/a					
Configuration File Keyword	alloc_prim_space *space*					√
STRUDM Parameter	n/a					

Values

space is equivalent to the first sub-parameter of the second sub-parameter of the JCL DD statement's SPACE parameter.

Valid values for space are any number (number of space units to allocate).

[Default is 15.]

References

Refer to the IBM JCL Reference manual for complete details.

6.16 ALLOC_RECFM

Description

The ALLOC_RECFM option is a dynamic allocation option that specifies the record format used for newly allocated data sets.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-alloc_recfm *format*					√
Environment Variable	n/a					
Configuration File Keyword	alloc_recfm *format*					√
STRUDM Parameter	n/a					

Values

format is equivalent to the JCL DD statement's **`RECFM`** parameter.

Valid values for *format* are dependent on the data set organization and access method used. The following values are valid for both partitioned and sequential data sets:

- **F**[**B**][**A**|**M**]
  Fixed, optionally blocked, and optionally either ANSI or Machine control characters.
- **V**[**B**][**A**|**M**|**S**]
  Variable, optionally blocked, and optionally either ANSI or Machine control characters, or spanned.

[Default is VB.]

References

Refer to the IBM JCL Reference manual for complete details.

6.17 ALLOC_SEC_SPACE

Description

The ALLOC_SEC_SPACE option is a dynamic allocation option that specifies the secondary space allocation used for newly allocated data sets.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-alloc_sec_space *space*					√
Environment Variable	n/a					
Configuration File Keyword	alloc_sec_space *space*					√
STRUDM Parameter	n/a					

Values

space is equivalent to the second sub-parameter of the second sub-parameter of the JCL DD statement's SPACE parameter.

Valid values for *space* are any number (number of space units to allocate).

[Default is 15.]

References

Refer to the IBM JCL Reference manual for complete details.

6.18 ALLOC_SPACE_UNIT

Description

The ALLOC_SPACE_UNIT option is a dynamic allocation option that specifies the space unit in which space is allocated for newly allocated data sets.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-alloc_space_unit *space*					√
Environment Variable	n/a					
Configuration File Keyword	alloc_space_unit *space*					√
STRUDM Parameter	n/a					

Values

space is equivalent to the first sub-parameter of the JCL DD statement's SPACE parameter.

Valid values for *space* are:

- *number*
  Block length or record length
- **cyl**
  Cylinder allocation
- **trk**
  Track allocation

[Default is trk.]

References

Refer to the IBM JCL Reference manual for complete details.

6.19 ALLOC_STORCLAS

Description

The ALLOC_STORCLAS option is a dynamic allocation option that specifies the SMS storage class used for newly allocated data sets.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-alloc_storclas *class*					√
Environment Variable	n/a					
Configuration File Keyword	alloc_storclas *class*					√
STRUDM Parameter	n/a					

Values

class is equivalent to the JCL DD statement's STORCLAS parameter.

Valid values for *class* are any SMS storage classes defined in the local environment.

[There is no default.]

References

Refer to the IBM JCL Reference manual for complete details.

6.20 ALLOC_UNIT

Description

The ALLOC_UNIT option is a dynamic allocation option that specifies the unit used for newly allocated data sets.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-alloc_unit *unit*					√
Environment Variable	n/a					
Configuration File Keyword	alloc_unit *unit*					√
STRUDM Parameter	n/a					

Values

unit is equivalent to the JCL DD statement's UNIT parameter.

Valid values for *unit* are:

- *number*
  Device number
- *name*
  Unit generic or group name

[Default is SYSALLDA.]

References

Refer to the IBM JCL Reference manual for complete details.

6.21 ALLOC_VOLSER

Description

The ALLOC_VOLSER option is a dynamic allocation option that specifies the volume serial number used for newly allocated data sets.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-alloc_volser *number*					√
Environment Variable	n/a					
Configuration File Keyword	alloc_volser *number*					√
STRUDM Parameter	n/a					

Values

number is equivalent to the sub-parameter SER of the JCL DD statement's VOL parameter.

Valid values for *number* are any Volume serial number.

[There is no default.]

References

Refer to the IBM JCL Reference manual for complete details.

6.22 BIF_DIRECTORY

Description

The BIF_DIRECTORY option specifies the Broker Interface File (BIF) directory where the Universal Broker interface file, **`ubroker.bif`**, is located.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-bif_directory *directory*			√		
Environment Variable	UDMBIFDIRECTORY=*directory*			√		
Configuration File Keyword	n/a					
STRUDM Parameter	n/a					

Values

directory is the name of the BIF directory.

[Default is `/var/opt/universal`.]

6.23 CA_CERTIFICATES

Description

The CA_CERTIFICATES option specifies the location of the PEM-formatted trusted Certificate Authority (CA) X.509 certificates file.

Trust CA certificates are required if Universal Broker certificate authentication and verification is desired.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-ca_certs *ddname* or *file*			√	√	√
Environment Variable	UDMCACERTS=*file*	√		√	√	
Configuration File Keyword	ca_certificates *ddname* or *file*	√		√	√	√
STRUDM Parameter	CACERTS(*file [lib]*) [CACERTSMBR (*member*)]	√				

Values

z/OS

ddname is the ddname of the X.509 certificates. The value is used only when the SSL_IMPLEMENTATION option is set to *OPENSSL*.

Allocated to the ddname must be either a sequential data set or a member of a PDS that has a variable record format.

UNIX and Windows

file is the path name of the X.509 certificates file. Relative paths are relative the current working directory.

IBM i

file is the qualified file name of the X.509 certificates file. The file name can be qualified by a library name. If not, the library list `*LIBL` is searched for the first occurrence of the file name.

6.24 CERTIFICATE

Description

The CERTIFICATE option specifies the file / ddname name of the PEM-formatted X.509 certificate that identifies the UDM Manager.

A UDM Manager X.509 certificate is required if the Universal Broker requires client authentication.

Note: If the CERTIFICATE option is used, the PRIVATE_KEY option also is required.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-cert *ddname* or *file*			√	√	√
Environment Variable	UDMCERT=*file*	√		√	√	
Configuration File Keyword	certificate *ddname* or *file*	√		√	√	√
STRUDM Parameter	CERT(*file [lib]*) [CERTMBR (*member*)]	√				

Values

z/OS

ddname is the ddname of the X.509 certificate. The value is used only when the SSL_IMPLEMENTATION option is set to *OPENSSL*.

Allocated to the ddname must be either a sequential data set or a member of a PDS that has a variable record format.

UNIX and Windows

file is the path name of the X.509 certificate file. Relative paths are relative to the current working directory.

IBM i

file is the qualified file name of the X.509 certificate file. The file name can be qualified by a library name. If not, the library list `*LIBL` is searched for the first occurrence of the file name.

6.25 CERTIFICATE_REVOCATION_LIST

Description

The CERTIFICATE_REVOCATION_LIST option specifies the file name / ddname of the PEM-formatted file containing the Certificate Revocation List (CRL) issued by the trusted Certificate Authority.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-crl *ddname* or *file*			√	√	√
Environment Variable	UDMCRL=*file*	√		√	√	
Configuration File Keyword	crl *ddname* or *file*	√		√	√	√
STRUDM Parameter	CRLFILE(*file [lib]*) [CRLMBR(*member*)]	√				

Values

z/OS

ddname is the ddname of the file containing the CRL. The value is used only when the SSL_IMPLEMENTATION option is set to *OPENSSL*.

UNIX and Windows

file is the path name of the file containing the CRL. Relative paths are relative to the current working directory.

IBM i

file is the qualified file name of the CRL file. The file name can be qualified by a library name. If not, the library list `*LIBL` is searched for the first occurrence of the file name.

6.26 CODE_PAGE

Description

The CODE_PAGE option specifies the character code page that is used to translate text data received and transmitted over the network.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	-t *codepage*			√	√	√
Command Line, Long Form	-codepage *codepage*			√	√	√
Environment Variable	UDMCODEPAGE=*codepage*	√		√	√	
Configuration File Keyword	codepage *codepage*	√		√	√	√
STRUDM Parameter	CODEPAGE(*codepage*)	√				

Values

codepage is the character code page that is used to translate data.

codepage references a Universal Translate Table (UTT) file provided with the product (see Section 21.13 UTT Files for information on UTT files). UTT files are used to translate between Unicode and the local single-byte code page. (All UTT files end with an extension of `.utt`.)

Note: UTF-8 is not a supported codepage value for CODE_PAGE. UTF-8 codepage is valid only for text file data translation. Consequently, it can be specified only on the UDM open script statement.

Default

IBM i

- IBM037

UNIX

- ISO8859-1 (8-bit ASCII) ASCII-based operating systems
- IBM1047 (EBCDIC) Non-IBM i, EBCDIC-based operating system

z/OS

- ISO8859-1 (8-bit ASCII) ASCII-based operating systems
- IBM1047 (EBCDIC) Non-IBM i, EBCDIC-based operating system

(See Section 21.12 Character Code Pages for a complete list of character code pages provided by Stonebranch Inc. for use with Stonebranch Solutions.)

6.27 CODEPAGE_TO_CCSID_MAP

Description

CAUTION: This option is intended only for use by IBM i specialists who fully understand code pages, CCSIDs, how the two relate to each other, and how IBM i uses CCSIDs for data translation between data streams and files. If a code page and CCSID are not correctly matched, data corruption will occur.

The CODEPAGE_TO_CCSID_MAP option specifies whether to use the internal table or external table for code page to CCSID mapping.

An internal table provides code page to CCSID mapping for the code page specified by the open command. The mapping only occurs if the CCSID is required for text file mapping. For the LIB file system, this includes mapping text to source physical files or to data files with an associated DDS file. All files in the root and QOpenSys file systems have associated CCSIDs.

This CCSID is not the same as the CCSID attribute associated with the attrib command; the UDM CCSID attribute determines the CCSID of the target file if the file does not exist. This CCSID in the mapping table is used as the CCSID associated with the attrib command when the default value, CODEPAGE, is specified; it also identifies the data stream CCSID, allowing the operating system to translate the code page translated data stream to the file. However, data written to or read from a file, record, or field with a CCSID of 65535 (or 'HEX') will not be translated.

The code page (and the mapped CCSID) from the open command is used for data mapping between the two parties involved in a data transfer. For IBM i, the code page also is used for mapping the data stream to or from the IBM i file, whether a LIB or HFS transfer.

Under normal circumstances, the external mapping table will not be needed. This external table replaces the internal table, so all potentially needed code page to CCSID mappings must be provided in the external table.

Stonebranch Solutions for IBM i provides an example external table in file **`CP2CCSID_X`**, in product library **`UNVPRD420`**. For UDM to use the external table, create a single member file with the name **`CDPG2CCSID`** in the installation library. The example file may be used as a template by copying it to **`CDPG2CCSID`** in the same library.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	n/a					
Environment Variable	n/a					
Configuration File Keyword	codepage_to_ccsid_map_opt *table*	√				
STRUDM Parameter	n/a					

Value

table is the specification for which table to use:

- **error**
  Use the external table; if it is not found, report an error.
- **quiet**
  Use the external table; if it is not found, use the internal table. No message is issued.
- **internal**
  Use the internal table.

[Default is internal.]

6.28 COMMENT

Description

The COMMENT option specifies a user-defined string that can contain any value.

This comment will appear for the server(s) in a transfer session.

Note: You also can create a comment for the servers in a single session (or override the comment specified by this option) via the open command.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-comment *text*			√	√	√
Environment Variable	n/a					
Configuration File Keyword	n/a					
STRUDM Parameter	COMMENT (*user-defined string*)	√				

Value

text is the user-defined string.

6.29 CTL_SSL_CIPHER_LIST

Description

The CTL_SSL_CIPHER_LIST option specifies the acceptable and preferred SSL cipher suites to use for the control session between UDM components.

The SSL protocol uses the cipher suites to specify which encryption and message authentication (or message digest) algorithms to use.

Usage

Method Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-ctl_ssl_cipher_list *list*			√	√	√
Environment Variable	UDMCTLSSLCIPHERLIST=*list*	√		√	√	
Configuration File Keyword	ctl_ssl_cipher_list *list*	√		√	√	√
STRUDM Parameter	CTLCPHRLST(*cipherlist*)	√				

Values

list is a comma-separated list of SSL cipher suites. The list should be ordered with the most preferred suite first and the least preferred suite last.

Valid *list* values are:

• RC4-SHA	128-bit RC4 encryption and SHA-1 message digest
• RC4-MD5	128-bit RC4 encryption and MD5 message digest
• AES256-SHA	256-bit AES encryption and SHA-1 message digest
• AES128-SHA	128-bit AES encryption and SHA-1 message digest
• DES-CBC3-SHA	128-bit Triple-DES encryption and SHA-1 message digest
• DES-CBC-SHA	128-bit DES encryption and SHA-1 message digest

[Default is RC4-SHA,RC4-MD5,AES256-SHA,AES128-SHA,DES-CBC3-SHA, DES-CBC-SHA]

6.30 DATA_COMPRESSION

Description

The DATA_COMPRESSION option specifies whether or not data in standard I/O file transmissions across the network should be compressed.

Optionally, it also can specify the compression method to use.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	-k *option[,method]*			√	√	√
Command Line, Long Form	-compress *option[,method]*			√	√	√
Environment Variable	UDMCOMPRESS=*option[,method]*	√		√	√	
Configuration File Keyword	compress *option[,method]*	√		√	√	√
STRUDM Parameter	COMPRESS(**option*) and CMPRSMTH(**method*)	√				

Values

option is either of the following values:

- **yes**
  Data compression is required. All data in standard I/O file transmissions is compressed regardless of the UDM Server DATA_COMPRESSION option value.
- **no**
  Data compression is not required. However, data compression still can be requested via the UDM Server DATA_COMPRESSION option.

[Default is no.]

method is either of the following values:

- **zlib**
  Data is compressed using ZLIB compression algorithm. This method usually results in a very high compression rate, but tends to be somewhat CPU-intensive. It is recommended in environments where controlling a process's CPU usage is not necessarily a priority.
- **hasp**
  Data is compressed using the HASP compression algorithm. This method is less CPU-intensive than the ZLIB method. It is recommended in environments where controlling CPU usage is a priority. With HASP, the compression rate, while still very good, tends to be a little less than what is possible with the ZLIB.

[Default is zlib.]

6.31 ENCRYPT

Description

The ENCRYPT option specifies the encryption method to be used in a UDM transfer session if one is not specified by an **`encrypt`** parameter in the UDM open command.

If an encryption method is specified by an **`encrypt`** parameter an open command, it overrides the method specified in ENCRYPT.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-encrypt *option*	√		√	√	√
Environment Variable	UDMENCRYPT	√		√	√	√
Configuration File Keyword	encrypt *option*	√		√	√	√
STRUDM Parameter	ENCRYPT(*option*)	√				

Values

option is either of the following values:

- **yes**
  An agreed-upon cipher will be negotiated based on the UDM Server DATA_SSL_CIPHER_LIST configuration option value.
- **no**
  NULL-MD5 is used as the encryption method.
- *cipher*
  Specific cipher to use as encryption method. Table 21.15 SSL Cipher Suites for UDM identifies the list of SSL cipher suites provided for UDM by Stonebranch Inc.

  Note:In order to establish a transfer session without using SSL for the data session, the NULL-NULL cipher must be specified in the cipher list for any UDM Server involved in the session and in the encrypt option of the open command.

[Default is no.]

6.32 DATA_SSL_CIPHER_LIST

Description

The DATA_SSL_CIPHER_LIST option specifies the acceptable and preferred SSL cipher suites to use for the data session on which file data is transferred between UDM primary and secondary servers.

The SSL protocol uses the cipher suites to specify which encryption and message authentication (or message digest) algorithms to use.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-data_ssl_cipher_list *list*			√	√	√
Environment Variable	UDMDATASSLCIPHERLIST=*list*	√		√	√	
Configuration File Keyword	data_ssl_cipher_list *list*	√		√	√	√
STRUDM Parameter	DTACPHRLST(*cipherlist*)	√				

Values

list is a comma-separated list of SSL cipher suites. The list should be ordered with the most preferred cipher suite first and the least preferred cipher suite last.

Table 21.15 SSL Cipher Suites for UDM identifies the list of SSL cipher suites provided for UDM by Stonebranch Inc.

Default

The default list of SSL cipher suites is:

RC4-SHA,RC4-MD5,AES256-SHA,AES128-SHA,DES-CBC3-SHA,DES-CBC-SHA, NULL-SHA,NULL-MD5

6.33 EVENT_GENERATION

Description

The EVENT_GENERATION option specifies which events are to be generated and processed as persistent events.

A persistent event record is saved in a Universal Enterprise Controller (UEC) database for long-term storage.

(For a list of all event types for all Stonebranch Solutions components, see the Universal Event Subsystem 4.2.0 Event Definitions document.)

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	n/a					
Environment Variable	n/a					
Configuration File Keyword	event_generation *types*	√		√	√	√
STRUDM Parameter	n/a					

Values

type specifies a comma-separated list of event types. It allows for all or a subset of all potential event message types to be selected.

Event type ranges can be specified by separating the lower and upper range values with a dash (**-**) character.

Event types can be selected for inclusion or exclusion:

- Inclusion operator is an asterisk (*****).
- Exclusion operator is **X** or **x**.

Examples

- 100,101,102
  Generate event types 100, 101, and 102.
- 100-102
  Generate event types 100 through 102.
- 100-102,200
  Generate event types 100 through 102 and 200.
- *
  Generate all event types.
- *,X100
  Generate all event types except for 100.
- x*
  Generate no event types.
- *,X200-250,X300
  Generate all event types except for 200 through 250 and 300.

[Default is *X (no event types).]**

6.34 FRAME_INTERVAL

Description

The FRAME_INTERVAL options sets the number of UDM transfer blocks transferred before a frame-sync message is sent when UDM is operating with network fault tolerance on (see NETWORK_FAULT_TOLERANT).

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-frame_interval *number*			√	√	√
Environment Variable	UDMFRAMEINTERVAL *number*	√		√	√	√
Configuration File Keyword	frame_interval *number*	√		√	√	√
STRUDM Parameter	FRAMEINT(*number*)	√				

Values

number can be any number.

[Default is *100*.]

Note: This value should not be changed without direction from Stonebranch, Inc. Customer Support. Changing this value could degrade UDM performance.

6.35 HELP

Description

The HELP option prints a description of the command options and their format.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	-h			√	√	√
Command Line, Long Form	-help			√	√	√
Environment Variable	n/a					
Configuration File Keyword	n/a					

Values

There are no values used with this option.

6.36 IDLE_TIMEOUT

Description

The IDLE_TIMEOUT option sets the number of seconds of inactivity in an interactive UDM session after which the UDM Manager will close the session.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-idle_timeout *seconds*			√	√	√
Environment Variable	UDMIDLETIMEOUT=*seconds*	√		√	√	√
Configuration File Keyword	idle_timeout *seconds*	√		√	√	√
STRUDM Parameter	IDLTIMOUT(*seconds*)	√				

Values

seconds can be any number.

[Default is *1200*.]

6.37 INSTALLATION_DIRECTORY

Description

The INSTALLATION_DIRECTORY option specifies the directory in which Universal Data Mover is installed.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	n/a					
Environment Variable	n/a					
Configuration File Keyword	installation_directory *directory*			√	√	
STRUDM Parameter	n/a					

Values

directory is any directory.

Defaults

UNIX

[Default is `/opt/universal/udmmgr`.]

Windows

[Default is UDM Manager installation file: `c:\Program Files\Universal\udmmgr`.]

6.38 KEEP_ALIVE_INTERVAL

Description

The KEEP_ALIVE_INTERVAL option sets the default interval (in seconds) at which a keep alive message is sent from the UDM Manager to the transfer server(s). If the transfer server(s) do not respond to the keep alive message within 3X this interval, a network fault is registered.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-keep_alive_interval *seconds*			√	√	√
Environment Variable	UDMKEEPALIIVEINTERVAL= *seconds*	√		√	√	√
Configuration File Keyword	keep_alive_interval *seconds*	√		√	√	√
STRUDM Parameter	KEEPALIVE(*seconds*)	√				

Values

seconds can be any number.

[Default is *120*.]

6.39 MERGE_LOG

Description

The MERGE_LOG option specifies whether or nor to merge standard out and standard error output streams from a remote command to the UDM transaction log (see Chapter 19 Remote Execution).

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	n/a					
Environment Variable	UDMMERGELOG=*option*	√		√	√	√
Configuration File Keyword	mergelog *option*	√		√	√	√
STRUDM Parameter	n/a					

Values

Valid *option* values are:

- **yes**
  Merge the standard out and standard error output streams.
- **no**
  Do not merge the standard out and standard error output streams.

[Default is no.]

6.40 MESSAGE_LANGUAGE

Description

The MESSAGE_LANGUAGE option specifies the Universal Message Catalog (UMC) file used to format messages.

UDM provides specific UMC files for specific languages. The first three characters of the language name are used as a three-character suffix in the UMC file base name, **UDMMC**. All UMC files then have a **.UMC** extension.

UMC files are located in the **UNVNLS** file in the Stonebranch Solutions installation library (default is **UNVPRD420**).

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	-L *language*			√	√	√
Command Line, Long Form	-lang *language*			√	√	√
Environment Variable	UDMLANG=*language*	√		√	√	
Configuration File Keyword	language *language*	√		√	√	√
STRUDM Parameter	MSGLANG(*language*)	√				

Values

language is any UMC file provided for UDM by Stonebranch Inc.

Note: For the current release of UDM, English is the only available language.

6.41 MESSAGE_LEVEL

Description

The MESSAGE_LEVEL option specifies the level of messages to write.

It also specifies, optionally, whether or not to write a date and time stamp with each message.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	-l *level[,time]*			√	√	√
Command Line, Long Form	-level *level[,time]*			√	√	√
Environment Variable	UDMLEVEL=*level[,time]*	√		√	√	
Configuration File Keyword	message_level *level[,time]*	√		√	√	√
STRUDM Parameter	MSGLEVEL(**level[*time]*)	√				

Values

level indicates either of the following level of messages:

- **trace**
  Activates tracing and generates a trace file to which UDM writes trace messages used for debugging.
  - UNIX Trace file (`udm.trc`) is created in the current working directory.
  - Z/OS Trace file is written to the UNVTRACE ddname.
  - IBM i Trace file name is `*CURLIB/UNVTRCUDM(Txxxxxx)` where `xxxxxx` is the process ID number of the job invoking Universal Data Mover.

  Note: Use **trace** only as directed by Stonebranch, Inc. Customer Support.
- **audit**
  Issues audit, informational, warning, and error messages.
- **info**
  Issues informational, warning, and error messages.
- **warn**
  Issues warning and error messages.
- **error**
  Issues error messages only.

IBM i and z/OS

[Default is info.]

UNIX and Windows

[Default is warn.]

time specifies either of the following:

- **time**
  Include a time and date stamp on each message.
- **notime**
  Do not include a time and date stamp on each message.

IBM i

time specifies either of the following:

- **yes**
  Include a time and date stamp on each message.
- **no**
  Do not include a time and date stamp on each message.

[Default is **yes**.]

z/OS

[Default is time.]

UNIX and Windows

[Default is notime.]

6.42 MODE_TYPE

Description

The MODE_TYPE option specifies the default transfer mode type for UDM sessions.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-mode_type *type*			√	√	√
Environment Variable	UDMMODETYPE=*type*	√		√	√	√
Configuration File Keyword	mode_type *type*	√		√	√	√
STRUDM Parameter	MODETYPE (**type*)	√				

Values

Valid *type* values are:

- **binary**
  Default transfer mode type is binary.
- **text**
  Default transfer mode type is text.

[Default is binary.]

IBM i

Valid *type* values are:

- **bin**
  Default transfer mode type is binary.
- **binary**
  Default transfer mode type is binary.
- **text**
  Default transfer mode type is text.

[Default is bin.]

6.43 NETWORK_DELAY

Description

The NETWORK_DELAY option sets the expected network latency (in seconds).

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-delay *seconds*			√	√	√
Environment Variable	UDMDELAY=*seconds*	√		√	√	√
Configuration File Keyword	network_delay *seconds*	√		√	√	√
STRUDM Parameter	DELAY(*seconds*)	√				

Values

seconds is any number.

[Default is *120*.]

6.44 NETWORK_FAULT_TOLERANT

Description

The NETWORK_FAULT_TOLERANT option sets whether or not UDM transfer sessions are network fault tolerant by default.

The Network Fault Tolerant (NFT) feature allows UDM to recover from network faults and continue processing without interruption. NFT is turned on or off with the UDM script open command.

NETWORK_FAULT_TOLERANT sets the default NFT value if it is not specified in the open command.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-network_fault_tolerant *option*			√	√	√
Environment Variable	UDMNETWORKFAULTTOLERANT= *option*	√		√	√	√
Configuration File Keyword	network_fault_tolerant *option*	√		√	√	√
STRUDM Parameter	NETWORKFT(**option*)	√				

Values

option can be either of the following values:

- **yes**
  Network fault tolerance is on for UDM sessions, allowing UDM to attempt to recover from a network fault and resume the session.
- **no**
  Network fault tolerance is off for UDM sessions.

[Default is yes.]

6.45 NLS_DIRECTORY

Description

The NLS_DIRECTORY option specifies the name of the directory where the UDM Manager message catalog and code page tables are located.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	n/a					
Environment Variable	n/a					
Configuration File Keyword	nls_directory *directory*			√	√	
STRUDM Parameter	n/a					

Values

directory can be any directory.

Full path names are recommended.

Relative path names are relative to the `universal` installation directory.

UNIX

[Default is `/opt/universal/nls`.]

Windows

[Default is `..\nls`.]

6.46 OPEN_RETRY

Description

The OPEN_RETRY option provides a level of fault tolerance for the **open** command.

If UDM cannot establish a transfer session due to network error — because a remote Broker was not running or, basically, for any reason other than an invalid user name or password — UDM will wait a period of time (specified by the OPEN_RETRY_INTERVAL option) and then retry to open a session.

UDM will attempt to establish a session until it is successful or it reaches the retry limit (as specified by the OPEN_RETRY_COUNT option). If the retry limit is reached, UDM will stop attempting to establish a session and will return an error.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	-open_retry *option*			√	√	√
Command Line, Long Form	-open_retry *option*			√	√	√
Environment Variable	UDMOPENRETRY=*option*	√		√	√	√
Configuration File Keyword	open_retry *option*	√		√	√	√
STRUDM Parameter	OPENRETRY (**enable count interval*)	√				

Values

Valid *option* values are:

- **yes**
  OPEN_RETRY used by UDM Manager.
- **no**
  OPEN_RETRY not used by UDM Manager.

[Default is no.]

IBM i

The STRUDM parameter (**OPENRETRY**) value contains three elements: *enable*, *count*, and *interval*. These are the same values specified by the OPEN_RETRY, OPEN_RETRY_COUNT, and OPEN_RETRY_INTERVAL options in the configuration file or when using environment variables.

Valid values for *enable* are:

- **yes**
  Enable open retry.
- **no**
  Disable open retry.

[Default is no.]

count is the maximum number of attempts that will be made to establish a session by the open command.

[Default is 20.]

interval is the number of seconds that UDM will wait between each open retry attempt.

[Default is 60.]

For example: `OPENRETRY(*yes 20 45)`.

6.47 OPEN_RETRY_COUNT

Description

The OPEN_RETRY_COUNT option sets the maximum number of attempts that will be made to establish a session by the open command.

This option is used only if the OPEN_RETRY option is being used by UDM (value=*yes*)

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-open_retry_count *number*			√	√	√
Environment Variable	UDMOPENRETRYCOUNT=*number*	√		√	√	√
Configuration File Keyword	open_retry_count *number*	√		√	√	√
STRUDM Parameter	n/a (see OPEN_RETRY option)					

Values

number is any number.

{Default is *20*.]

6.48 OPEN_RETRY_INTERVAL

Description

The OPEN_RETRY_INTERVAL option sets the number of seconds that UDM will wait between each open retry attempt (see OPEN_RETRY_COUNT).

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-open_retry_interval *number*			√	√	√
Environment Variable	UDMOPENRETRYINTERVAL= *number*	√		√	√	√
Configuration File Keyword	open_retry_interval *number*	√		√	√	√
STRUDM Parameter	n/a (see OPEN_RETRY option)					

Values

number is any number (of seconds).

[Default is *60*.]

6.49 OUTBOUND_IP

Description

The OUTBOUND_IP option sets the host or IP address that UDM binds to when initiating outgoing connections to another UDM server.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-outboundip *host*			√	√	√
Environment Variable	UDMOUTBOUNDIP=*host*	√		√	√	√
Configuration File Keyword	outbound_ip *host*	√		√	√	√
STRUDM Parameter	OUTBOUNDIP (*host*)	√				

Values

host is the required IP address.

[There is no default.]

6.50 PLF_DIRECTORY

Description

The PLF_DIRECTORY option specifies the Program Lock File (PLF) directory where the program lock files are located.

A program lock file is created and used by the UDM Manager process to store manager process termination information for the Universal Broker.

IBM i

Do not include this directory in any system or backup that requires an exclusive lock on the directory while UDM is running.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-plf_directory *directory*			√		
Environment Variable	UDMPLFDIRECTORY=*directory*			√		
Configuration File Keyword	n/a					
STRUDM Parameter	PLFDIR (*directory*)	√				

Values

directory is the name of the PLF directory.

A full path name must be specified.

Defaults

UNIX

[Default is `/var/opt/universal/tmp`.]

IBM i

[Default is `/tmp`.]

6.51 PRIVATE_KEY

Description

The PRIVATE_KEY option specifies the location of the PEM-formatted RSA private key that corresponds to the X.509 certificates specified by the CERTIFICATE option.

Note: PRIVATE_KEY is required only if a certificate is specified by CERTIFICATE.

z/OS

PRIVATE_KEY is used only when the SSL_IMPLEMENTATION option is set to *OPENSSL*.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-private_key *ddname* or *file*			√	√	√
Environment Variable	UDMPRIVATEKEY=*file*	√		√	√	
Configuration File Keyword	private_key *ddname* or *file*	√		√	√	√
STRUDM Parameter	PVTKEYF(*file [lib]*) [PVTKEYFMBR (*member*)]	√				

Values

z/OS

ddname is the ddname of the PEM-formatted RSA private key that corresponds to the X.509 certificates.

Allocated to the ddname must be either a sequential data set or a member of a PDS that has a variable record format.

UNIX and Windows

file is the path of the PEM-formatted RSA private key file that corresponds to the X.509 certificates.

IBM i

file is the qualified name of the PEM-formatted RSA private key file that corresponds to the X.509 certificates.

The file name can be qualified by a library name. If not, the library list `*LIBL` is searched for the first occurrence of the file name.

6.52 PRIVATE_KEY_PWD

Description

The PRIVATE_KEY_PWD option specifies the password or passphrase for the PEM-formatted RSA private key specified with the PRIVATE_KEY option.

Note: Whether or not the password is required or not depends on whether or not it is required by the private key.

z/OS

PRIVATE_KEY_PWD is used only when the SSL_IMPLEMENTATION option is set to *OPENSSL*.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-private_key_pwd *password*			√	√	√
Environment Variable	UDMPRIVATEKEYPWD=*password*	√		√	√	
Configuration File Keyword	private_key_password=*password*	√		√	√	√
STRUDM Parameter	PVTKEYPWD (*password*)	√				

Values

password is the password for the private key.

IBM i

Characters may be incorrectly translated due to reverse representations under 037 and 1047 CCSIDs:

• hat (circumflex)	logical not
• left bracket	Y acute
• right bracket	diaeresis (umlaut)

The hex/decimal exchanges are:

• 5F/95	B0/176
• AD/173	BA/186
• BD/189	BB/187

6.53 PROXY_CERTIFICATES

Description

The PROXY_CERTIFICATES option specifies whether or not UDM will use the managers certificate in a three-party transfer session if a certificate is supplied to the UDM Manager.

Proxy certificates are used only for three-party transfer sessions. All components, manager, primary and secondary, must be version 3.2 or later and must be using OpenSSL (System SSL does not support proxy certificates).

If PROXY_CERTIFICATES is set to yes, the UDM Manager's certificate is used to create a proxy certificate for the primary to use when connecting to the secondary. The proxy certificate has the same subject name as the managers certificate, so the certificate ACL for the secondary can be set up to look just like the primary's ACL.

Note: For more information on X509 proxy certificates, see the RFC at:

http://www.globus.org/alliance/publications/papers/pki04-welch-proxy-cert-final.pdf

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-proxy_certificates *option*			√	√	√
Environment Variable	UDMPROXYCERTIFICATES=*option*	√		√	√	
Configuration File Keyword	proxy_certificates *option*	√		√	√	√
STRUDM Parameter	PROXYCERT (*option*)	√				

Values

option is the specification for whether or not UDM will use proxy certificates.

Valid values for *option* are:

- **yes**
  UDM will use proxy certificates.
- **no**
  UDM will not use proxy certificates.

[Default is no.]

6.54 RECONNECT_RETRY_COUNT

Description

The RECONNECT_RETRY_COUNT option sets the number of attempts that the UDM Manager will make to re-establish a transfer session when a network fault occurs.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-retry_count *number*			√	√	√
Environment Variable	UDMRETRYCOUNT=*number*	√		√	√	√
Configuration File Keyword	reconnect_retry_count *number*	√		√	√	√
STRUDM Parameter	RETRYCNT(*number*)	√				

Values

number is any number.

{Default is *20*.]

6.55 RECONNECT_RETRY_INTERVAL

Description

The RECONNECT_RETRY_INTERVAL option sets the number of seconds that UDM will wait between each successive attempt to reestablish a transfer session when a network fault occurs.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-retry_interval *seconds*			√	√	√
Environment Variable	UDMRETRYINTERVAL=*seconds*	√		√	√	√
Configuration File Keyword	reconnect_retry_interval *seconds*	√		√	√	√
STRUDM Parameter	RETRYINT(*seconds*)	√				

Values

seconds is any number of seconds to wait.

[Default is *60*.]

6.56 RECV_BUFFER_SIZE

Description

The RECV_BUFFER_SIZE option sets the size (in bytes) of the TCP receive buffer for UDM.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-recvbuffersize *size*			√	√	√
Environment Variable	UDMRECVBUFSSIZE=*size*	√		√	√	√
Configuration File Keyword	recv_buffer_size *size*	√		√	√	√
STRUDM Parameter	RCVBUFSIZE(*size*)	√				

Values

size is the number of bytes.

[Default is *0*.]

(The default, **0,** instructs UDM to use the operating system default.)

Note: The size of the TCP receive buffer should be changed only when performance tweaking is necessary. Changing this value could affect performance adversely.

6.57 REMOTE_PORT

Description

The REMOTE_PORT option specifies the TCP port on the remote computer used for invoking UDM Server instances.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	-p *port*			√	√	√
Command Line, Long Form	-port *port*			√	√	√
Environment Variable	UDMPORT=*port*	√		√	√	
Configuration File Keyword	port *port*	√		√	√	√
STRUDM Parameter	PORT(*port*)	√				

Values

port is the TCP port on the remote computer.

Valid values for *port* are:

- Number
- Service name (for example, **ubroker**)

[Default is *7887*.]

6.58 SAF_KEY_RING

Description

The SAF_KEY_RING option specifies the SAF (RACF is a SAF implementation) certificate key ring name that the Universal Data Mover job should used for its certificate.

The key ring must be associated with the user profile with which the Universal Data Mover job executes.

Note: SAF_KEY_RING is required if the SSL_IMPLEMENTATION option is set to **system**.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-saf_key_ring *name*					√
Environment Variable	UDMSAFKEYRING=*name*					√
Configuration File Keyword	saf_key_ring *name*					√
STRUDM Parameter	n/a					

Values

name is the name of the SAF certificate key ring.

6.59 SAF_KEY_RING_LABEL

Description

The SAF_KEY_RING_LABEL option specifies the label of the certificate in the SAF (RACF is a SAF implementation) certificate key ring that the Universal Data Mover job should use for its certificate.

(The key ring is specified by the SAF_KEY_RING option.)

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-saf_key_ring_label *label*					√
Environment Variable	UDMSAFKEYRINGLABEL=*label*					√
Configuration File Keyword	saf_key_ring_label *label*					√
STRUDM Parameter	n/a					

Values

label is the label of the SAF certificate key ring.

[Default is the default certificate in the key ring.]

6.60 SCRIPT

Description

The SCRIPT option specifies the ddname from which to read a UDM script command file.

Note: This option overrides the default **`UNVSCR`** ddname.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	-s *ddname*					√
Command Line, Long Form	-script *ddname*					√
Environment Variable	n/a					
Configuration File Keyword	n/a					
STRUDM Parameter	n/a					

Values

ddname is the ddname from which to read the file.

6.61 SCRIPT_FILE

Description

The SCRIPT_FILE option specifies a script file containing UDM commands to execute.

UDM requires a member name; however, the value *FILE is valid and provides the IBM i default file search order.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	-s *filename*			√	√	
Command Line, Long Form	-script *filename*			√	√	
Environment Variable	UDMSCRIPT=*filename*	√		√	√	
Configuration File Keyword	n/a					
STRUDM Parameter	SCRFILE([*library*/] *filename*) [SCRMBR (*member*)]	√				

Values

filename is the name of the file from which the script is read.

6.62 SCRIPT_OPTIONS

Description

The SCRIPT_OPTIONS option specifies options to pass to the script command file.

Note: This option is valid only if a script file is specified.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	-o *options*			√	√	√
Command Line, Long Form	-options *options*			√	√	√
Environment Variable	UDMOPTIONS=*options*	√		√	√	
Configuration File Keyword	n/a					
STRUDM Parameter	OPTIONS (*options*)	√				

Values

options is the name of the options to pass to the script file.

IBM i, UNIX, and Windows

If *options* contain spaces, it must be enclosed in double (**"**) quotation marks. If a quotation mark is part of the value, prefix it with the Windows escape character, backslash (\).

z/OS

The format of *options* is the same as the UDM script call command.

6.63 SEND_BUFFER_SIZE

Description

The SEND_BUFFER_SIZE option sets the size (in bytes) of the TCP send buffer for UDM.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-sendbuffersize *size*			√	√	√
Environment Variable	UDMSENDBUFSSIZE=*size*	√		√	√	√
Configuration File Keyword	Send_buffer_size *size*	√		√	√	√
STRUDM Parameter	SNDBUFSIZE (*size*)	√				

Values

size is the number of bytes.

[Default is *0*.]

(The default, **0,** instructs UDM to use the operating system default.)

Note: The size of the TCP send buffer should be changed only when performance tweaking is necessary. Changing this value could affect performance adversely.

6.64 SERVER_STOP_CONDITIONS

Description

The SERVER_STOP_CONDITIONS option specifies one or more exit codes of the executing UDM Manager that should trigger the locally running Universal Broker to cancel the corresponding UDM Server for the exited UDM Manager.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-server_stop_conditions *codes*					√
Environment Variable	UDMSERVERSTOPCONDITIONS= *codes*					√
Configuration File Keyword	server_stop_conditions *codes*					√
STRUDM Parameter	n/a					

Values

codes is an exit code, or a comma-separated list of exit codes, that should cause the UDM Server to be cancelled.

z/OS ABEND codes are specified in two different formats:

• System ABEND code	Starts with S followed by a 3-character hexadecimal value.
• User ABEND code	Starts with U followed by a 4-character decimal value.

For example, when a job is terminated with the CANCEL console command, the job ends with a system ABEND code of S222.

[There is no default.]

6.65 SIZE_ATTRIB

Description

The SIZE_ATTRIB option sets the default size (number of records) for file creation of physical files for both data and source file types.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	n/a					
Environment Variable	UDMSIZEATTRIB=*size*	√				
Configuration File Keyword	size_attrib *size*	√				
STRUDM Parameter	n/a					

Values

size is the number of records.

Other than the leading * for *NOMAX, it can be any valid value. (To specify *NOMAX, enter 'NOMAX'.)

The values for Initial Number of Records, Increment Number of Records, and Maximum increments are comma-separated, not space-separated.

Examples:

- 1000,100,100
- 10000,499
- 50000

UDM uses this value as the default file SIZE attribute. (See Table 18.2 IBM i-Specific LIB File Attributes for Creating New Files for details.)

[default is *empty string*].

6.66 SSL_IMPLEMENTATION

Description

The SSL_IMPLEMENTATION option specifies the Secure Socket Layer (SSL) implementation to be used for network communications.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-ssl_implementation *option*					√
Environment Variable	UDMSSLIMPLEMENTATION=*option*					√
Configuration File Keyword	ssl_implementation *option*					√
STRUDM Parameter	n/a					

Values

option is the SSL implementation to be used.

Valid values for option are:

- **openssl**
  OpenSSL SSL library is used for the SSL protocol.
- **system**
  z/OS System SSL library is used for the SSL protocol. The z/OS System SSL library has installation and configuration prerequisites. (See the Stonebranch Solutions 4.2.0 Installation Guide for a description of the prerequisites before using System SSL.)

[Default is openssl.]

6.67 SYSTEM_ID

Description

The SYSTEM_ID option identifies the local Universal Broker with which the UDM Manager must register before the Manager performs any request.

Each Universal Broker running on a system is configured with a system identifier that uniquely identifies the Broker.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-system_id *ID*					√
Environment Variable	UDMSYSTEMID=*ID*					√
Configuration File Keyword	n/a					
STRUDM Parameter	n/a					

Values

ID is the system identifier of the local Universal Broker.

References

Refer to the local Universal Broker administrator for the appropriate system ID to use.

6.68 TCP_NO_DELAY

Description

The TCP_NO_DELAY option specifies whether or not to use TCP packet coalescing.

The packet coalescing algorithm, which can delay the sending of small amounts of data over the network, is designed to improve network congestion. However, since it can have a significantly negative effect on the performance of UDM, TCP_NO_DELAY specifies – by default – not to use TCP packet coalescing.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-tcp_no_delay *option*	√		√	√	√
Environment Variable	UDMTCPNODELAY=*option*			√	√	√
Configuration File Keyword	tcp_no_delay *option*	√		√	√	√
STRUDM Parameter	TCPNODELAY(*option*)	√				

Values

option is the specification for whether or not to use TCP packet coalescing.

Valid values for *option* are:

- **yes**
  Do not use TCP packet coalescing.
- **no**
  Use TCP packet coalescing.

[Default is yes.]

6.69 TRACE_FILE_LINES

Description

The TRACE_FILE_LINES option specifies the maximum number of lines to write to the trace file.

When the maximum number of lines has been reached, the trace file will wrap around and start writing trace entries after the trace header lines.

Tracing is activated, and a trace file is generated, when the MESSAGE_LEVEL option is set to **trace**.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-tracefilelines *number*			√	√	√
Environment Variable	UDMTRACEFILELINES=*number*	√		√	√	
Configuration File Keyword	trace_file_lines *number*	√		√	√	√
STRUDM Parameter	TRCLINES(*number*)	√				

Values

number is the maximum number of lines.

[Default is *200,000*.]

z/OS

Note: This option has no effect when the UNVTRACE ddname allocates a JES SYSOUT data set.
The average size of a trace file line is 50 characters.

UNIX

The average size of a trace file line is 50 characters.

IBM i

The current record length is 384 characters.

As allocated on the AS/400 (i5), the maximum number of records is 509,000. To increase the value of TRACE_FILE_LINES beyond this number, a manual adjustment of the file size via the CHGPF command is required.

Note:

1. The file is not created until the first time Universal Data Mover Manager trace is used.
2. Deleting the trace file will reset the maximum number of records to 509,000.
3. To clear the trace file without resetting the maximum number of records that the file may contain, use the **`RMVM FILE(*CURLIB/UNVTRCUDM) MBR(*ALL)`** command. Substitute an appropriate library for **`*CURLIB`** if needed.

Note: Setting a number of records larger than the maximum number of records allocated to the trace file will result in a prompt sent to **`QSYSOPR`**. Unless a default response is set for this message (CPA5305), the PROCESS WILL HANG waiting for the **`QSYSOPR`** response.

6.70 TRACE_TABLE

Description

The TRACE_TABLE option specifies the size of a wrap-around trace table maintained in memory.

The trace table is written when the program ends under the conditions specified by this option.

Tracing is activated, and a trace file is generated, when the MESSAGE_LEVEL option is set to **trace**.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-trace_table *size,condition*			√	√	√
Environment Variable	UDMTRACETABLE=*size,condition*	√		√	√	
Configuration File Keyword	trace_table *size,condition*	√		√	√	√
STRUDM Parameter	TRCTBL(*size, *condition)*	√				

Values

size is the size (in bytes) of the table. (A value of *0* indicates that the trace table is not used.)

The size can be suffixed with either of the following characters:

- **M** indicates that the size is specified in megabytes
- **K** indicates that the size is specified in kilobytes

For example, **50M** indicates that 50 X 1,048,576 bytes of memory is allocated for the trace table.

condition is the condition under which the trace table is written.

Possible values for *condition* are:

- **error**
  Write the trace table if the program ends with a non-zero exit code.
- **always**
  Write the trace table when the program ends regardless of the exit code.
- **never**
  Never write the trace table.

6.71 UCMD_PATH

Description

The UCMD_PATH option sets the complete path to Universal Command (UCMD) for calls by the **`exec`** command.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	n/a					
Environment Variable	UDMUCMDPATH=*path*			√	√	
Configuration File Keyword	ucmd_path *path*			√	√	
STRUDM Parameter	n/a					

Values

path must contain the UCMD file itself (for example: **`/opt/universal/bin/ucmd`**).

By default, the UCMD_PATH option is not set; UDM uses the **`PATH`** environment variable to call UCMD.

6.72 UMASK

Description

The UMASK option specifies the file mode creation mask. It affects the file permission bits of newly created files.

All files are created with a permission mode of 666, which is read-write permission for the owner, group, and other permission categories. UDM uses UMASK to turn off selected permission bits by subtracting the value of UMASK from mode 666.

IBM i

UMASK applies only to the **HFS** file system.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	-umask *number*			√	√	√
Environment Variable	UDMUMASK=*number*	√		√	√	√
Configuration File Keyword	umask *number*	√		√	√	√
STRUDM Parameter	n/a					

Values

number can be any number, 001 to 666.

[Default is *026*.]

The default value (**026**) results in file permission 640 (666 - 026 = 640), which is:

- read-write for the owner
- read for the group
- none for others

References

Refer to the UNIX man page umask(1) for complete details.

6.73 USAP_PATH

Description

The USAP_PATH option sets the complete path to USAP for calls by the **execsap** command.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	n/a					
Command Line, Long Form	n/a					
Environment Variable	UDMUSAPDPATH=*path*			√		
Configuration File Keyword	usap_path *path*			√		
STRUDM Parameter	n/a					

Values

path must contain the USAP file itself (for example: **/opt/universal/bin/usap**).

By default, the USAP_PATH option is not set; UDM uses the **PATH** environment variable to call USAP.

6.74 VERSION

Description

The VERSION option writes the program version information and copyright.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Command Line, Short Form	-v			√	√	√
Command Line, Long Form	-version			√	√	√
Environment Variable	n/a					
Configuration File Keyword	n/a					
STRUDM Parameter	VERSION(**option*)	√				

Values

There are no values used with this option.

IBM i

Valid values for *option* are:

- **yes**
  Write the program version information and copyright.
- **no**
  Do not write the program version information and copyright.

CHAPTER 7

Universal Data Mover Server for z/OS

7.1 Overview

This chapter provides the following information on the Universal Data Mover (UDM) Server, specific to the z/OS operating system:

- Component Definition
- Configuration

7.2 Component Definition

All Stonebranch Solutions components managed by Universal Broker have a component definition. The component definition is a text file of options containing component-specific information required by Universal Broker. (For details on how Universal Broker manages components, see the Universal Broker Reference Guide.)

The UDM Server for z/OS component definition is located in the component definition library **#HLQ.UNV.UNVCOMP** allocated to the Universal Broker ddname **UNVCOMP**. The UDM Server component definition member is **UDSCMP00**.

Table 7.1, below, identifies all of the options that comprise the UDM Server for z/OS component definition.

Each **Option Name** is a link to detailed information about that component definition option.

Option Name	Description
AUTOMATICALLY_START	Specification for whether or not UDM Server starts automatically when Universal Broker is started.
COMPONENT_NAME	Name by which the clients know the UDM Server.
CONFIGURATION_FILE	Member name of the UDM Server configuration file in the **UNVCONF** library allocated to the Broker ddname **UNVCONF**.
RUNNING_MAXIMUM	Maximum number of UDM Servers that can run simultaneously.
START_COMMAND	Member name of the UDM Server program.
WORKING_DIRECTORY	HMS directory used as the working directory of the UDM Server.

Table 7.1 UDM Server for z/OS - Component Definition Options

7.3 Configuration

Universal Data Mover Server configuration consists of defining runtime and default values.

7.3.1 Configuration File

The configuration file provides the simplest method of specifying configuration values that will not change with each command invocation.

The UDM Server configuration file name is specified in the UDM Server component definition. The default name is **`UDSCFG00`**. The name refers to a member in the PDS allocated to the Universal Broker ddname UNVCONF.

7.3.2 Configuration Options

Figure 7.2, below, identifies all UDM Server for z/OS command options.

Each **Option Name** is a link to detailed information about that command option.

Option	Description
ACTIVITY_MONITORING	Specification for whether or not product activity monitoring events are generated.
ALLOC_ABNORMAL_DISP	Abnormal disposition of a data set being allocated.
ALLOC_BLKSIZE	Block size used for newly allocated data sets.
ALLOC_DATACLAS	SMS data class used for newly allocated data sets.
ALLOC_DIR_BLOCKS	Number of directory blocks for newly allocated partitioned data sets.
ALLOC_DSORG	Data set organization used for newly allocated data sets.
ALLOC_INPUT_STATUS	Status of data sets being allocated for input.
ALLOC_LRECL	Logical record length used for newly allocated data sets.
ALLOC_MGMTCLAS	SMS management class used for newly allocated data sets.
ALLOC_NORMAL_DISP	Normal disposition of a data set being allocated.
ALLOC_OUTPUT_STATUS	Status of existing data sets being allocated for output.
ALLOC_PRIM_SPACE	Primary space allocation used for newly allocated data sets.
ALLOC_RECFM	Record format used for newly allocated data sets.
ALLOC_SEC_SPACE	Secondary space allocation used for newly allocated data sets.
ALLOC_SPACE_UNIT	Space unit in which space is allocated for newly allocated data sets.
ALLOC_STORCLAS	SMS storage class used for newly allocated data sets.
ALLOC_UNIT	Unit used for newly allocated data sets.
ALLOC_VOLSER	Volume serial number used for newly allocated data sets.
CODE_PAGE	Character code page used to translate text data.
DATA_COMPRESSION	Specification for whether or not data is compressed on all standard I/O files.
DATA_SSL_CIPHER_LIST	SSL cipher suites to use for data session between UDM primary and secondary servers.
EVENT_GENERATION	Events to be generated as persistent events.
FRAME_INTERVAL	Number of UDM transfer blocks transferred before a frame-sync message is sent with network fault tolerance turned on.
MESSAGE_LEVEL	Level of messages that UDM will write to the Universal message Catalog (UMC) file.
NETWORK_DELAY	Expected network latency (in seconds).
OUTBOUND_IP	Host or IP address that UDM binds to when initiating outgoing connections to another UDM server.
RECONNECT_RETRY_COUNT	Number of attempts that the UDM Manager will make to re-establish a transfer session when a network fault occurs.

Option	Description
RECONNECT_RETRY_INTERVAL	Number of seconds that UDM will wait between each successive attempt to re-establish a transfer session when a network fault occurs.
RECV_BUFFER_SIZE	Size of the TCP receive buffer for UDM.
SEND_BUFFER_SIZE	Size of the TCP send buffer for UDM.
TCP_NO_DELAY	Specification for whether or not to use TCP packet coalescing.
TMP_DIRECTORY	Directory that UDM Server uses for temporary files.
TRACE_FILE_LINES	Maximum number of lines to write to the trace file.
TRACE_TABLE	Size of a wrap-around trace table maintained in memory.
UMASK	File mode creation mask.
USER_SECURITY	User security option.

Table 7.2 UDM Server for z/OS - Configuration Options

7.4 Universal Access Control List

UDM Server uses the Universal Access Control List (UACL) file as an extra layer of security. The UACL file contains Universal Data Mover Server entries that contain Access Control List (ACL) rules that permit or deny access to the Server.

7.4.1 UACL Entries

The syntax of a UACL entry file is the same as the UDM configuration file.

Table 7.3 identifies all UDM Server for z/OS UACL entries.

Each **UACL Entry Name** is a link to detailed information about that UACL entry.

UACL Entry Name	Description
UDM_ACCESS	Allows or denies access to Universal Data Mover Server services
UDM_MGR_ACCESS	Allows or denies access based on the host name and/or user of the Manager trying to initiate a UDM session

Table 7.3 UDM Server for z/OS - UACL Entries

CHAPTER 8

Universal Data Mover Server for Windows

8.1 Overview

This chapter provides the following information on the Universal Data Mover (UDM) Server, specific to the Windows operating system:

- Component Definition
- Configuration
- Universal Access Control List

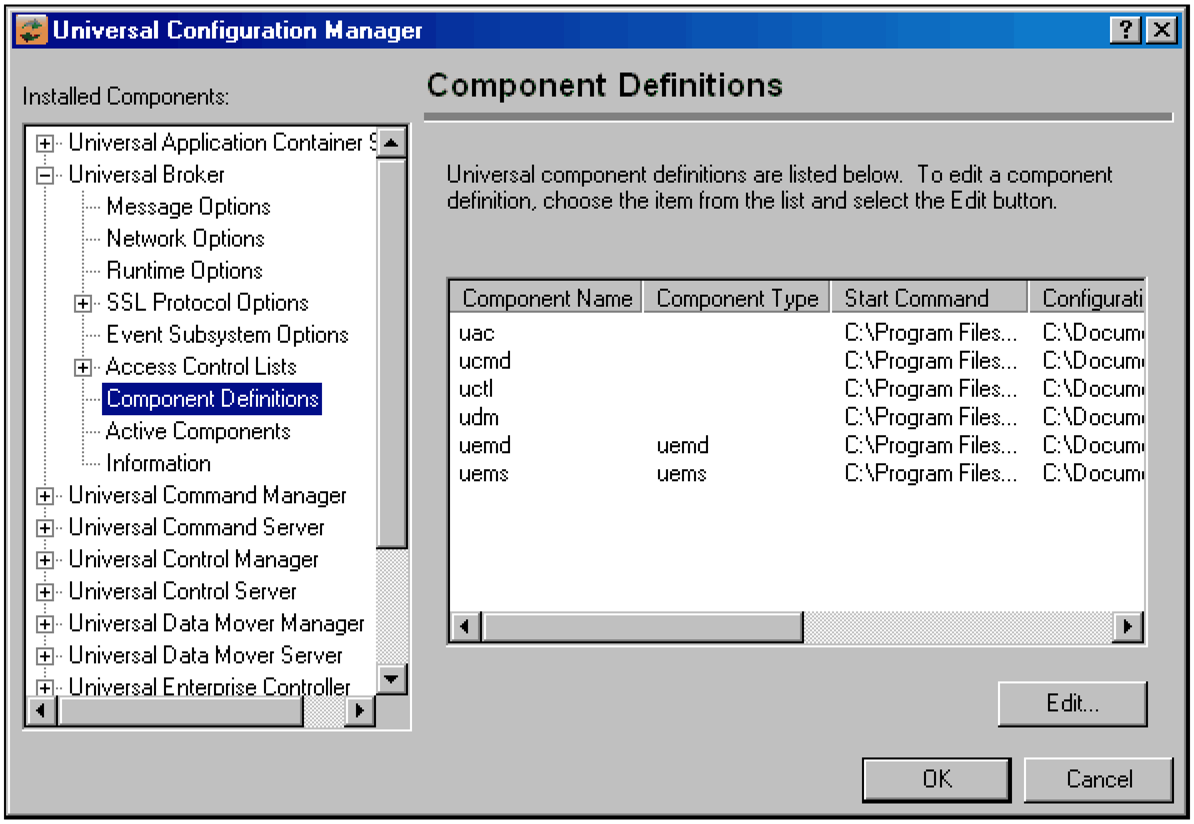
8.2 Component Definition

All Stonebranch Solutions components managed by Universal Broker have a component definition. The component definition is a text file of options containing component-specific information required by Universal Broker. (For details on how Universal Broker manages components, see the Universal Broker Reference Guide.)

The syntax of a component definition file is the same as a configuration file.

Although component definition files can be edited with any text editor (for example, Notepad), the Universal Configuration Manager application is the recommended way to edit component definitions for Windows.

Note: The component definitions for all Stonebranch Solutions components are identified in the Component Definitions property page of the Universal Broker (see Figure 8.1, below).



Universal component definitions are listed below. To edit a component definition, choose the item from the list and select the Edit button.

Component Name	Component Type	Start Command	Configurati
uac		C:\Program Files...	C:\Docum
ucmd		C:\Program Files...	C:\Docum
uctl		C:\Program Files...	C:\Docum
udm		C:\Program Files...	C:\Docum
uemd	uemd	C:\Program Files...	C:\Docum
uems	uems	C:\Program Files...	C:\Docum

Figure 8.1 Universal Configuration Manager - Component Definitions

The UDM Server component definition is located in the component definition directory of the Universal Broker.

Table 8.1, below, identifies all of the options that comprise the UDM for Windows component definition.

Each **Option Name** is a link to detailed information about that component definition option.

Option Name	Description
AUTOMATICALLY_START	Specification for whether or not UDM Server starts automatically when Universal Broker is started.
COMPONENT_NAME	Name by which the clients know the UDM Server.
CONFIGURATION_FILE	Full path name of the UDM Server configuration file.
RUNNING_MAXIMUM	Maximum number of UDM Servers that can run simultaneously.
START_COMMAND	Full path name of the UDM Server program.
WORKING_DIRECTORY	Full path name of the UDM Server working directory.

Table 8.1 UDM Server for Windows - Component Definition Options

8.3 Configuration

Universal Data Mover Server configuration consists of defining runtime and default values. This section describes the UDM Server configuration options.

8.3.1 Configuration File

The configuration file provides a method of specifying configuration values that will not change with each command invocation.

The Universal Data Mover Server configuration file name (and directory) is specified in the Universal Data Mover Server component definition (see Chapter 12 Universal Data Mover Component Definition Options). The default configuration file name is **`udms.conf`**.

Although configuration files can be edited manually with any text editor (for example, Notepad), the Universal Configuration Manager application is the recommended way to set configuration options in the configuration file.

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.

8.3.2 Configuration Options

Table 8.2 identifies all Universal Data Mover Server for Windows configuration options.

Each **Option Name** is a link to detailed information about that configuration option.

Option	Description
ACTIVITY_MONITORING	Specification for whether or not product activity monitoring events are generated.
CODE_PAGE	Character code page used to translate text data.
DATA_COMPRESSION	Specification for whether or not data is compressed on all standard I/O files.
DATA_SSL_CIPHER_LIST	SSL cipher suites to use for data session between UDM primary and secondary servers.
EVENT_GENERATION	Events to be generated as persistent events.
FRAME_INTERVAL	Number of UDM transfer blocks transferred before a frame-sync message is sent with network fault tolerance turned on.
INSTALLATION_DIRECTORY	Directory on which UDM Server is installed.
LOGON_METHOD	Specification for how users are logged onto the system.
MESSAGE_LEVEL	Level of messages that UDM will write to the Universal message Catalog (UMC) file.
NETWORK_DELAY	Expected network latency (in seconds).
NLS_DIRECTORY	Directory where the UDM Manager message catalog and code page tables are located.
OUTBOUND_IP	Host or IP address that UDM binds to when initiating outgoing connections to another UDM server.
RECONNECT_RETRY_COUNT	Number of attempts that the UDM Manager will make to re-establish a transfer session when a network fault occurs.
RECONNECT_RETRY_INTERVAL	Number of seconds that UDM will wait between each successive attempt to re-establish a transfer session when a network fault occurs.
RECV_BUFFER_SIZE	Size of the TCP receive buffer for UDM.
SEND_BUFFER_SIZE	Size of the TCP send buffer for UDM.
TCP_NO_DELAY	Specification for whether or not to use TCP packet coalescing.
TMP_DIRECTORY	Directory that UDM Server uses for temporary files.
TRACE_DIRECTORY	Directory name that UDM Server uses for its Trace files.
TRACE_FILE_LINES	Maximum number of lines to write to the trace file.
TRACE_TABLE	Size of a wrap-around trace table maintained in memory.
USER_SECURITY	User security option.

Table 8.2 UDM Server for Windows - Configuration Options

8.4 Universal Access Control List

UDM Server uses the Universal Access Control List (UACL) file as an extra layer of security. The UACL file contains UDM Server entries that contain Access Control List (ACL) rules that permit or deny access to the UDM Server.

8.4.1 UACL Entries

The syntax of a UACL entry file is the same as the UDM configuration file.

Table 8.3 identifies all Universal Data Mover Server for Windows UACL entries.

Each **UACL Entry Name** is a link to detailed information about that UACL entry.

UACL Entry Name	Description
UDM_ACCESS	Allows or denies access to Universal Data Mover Server services
UDM_MGR_ACCESS	Allows or denies access based on the host name and/or user of the Manager trying to initiate a UDM session

Table 8.3 UDM Server for Windows - UACL Entries

CHAPTER 9

Universal Data Mover Server for UNIX

9.1 Overview

This chapter provides the following information on Universal Data Mover (UDM) Server, specific to the UNIX operating system:

- Component Definition
- Configuration
- Universal Access Control List

9.2 Component Definition

All Stonebranch Solutions components managed by Universal Broker have a component definition. The component definition is a text file of options containing component-specific information required by Universal Broker. (For details on how Universal Broker manages components, see the Universal Broker Reference Guide.)

The syntax of a component definition file is the same as a configuration file.

The UDM Server for UNIX component definition is located in the component definition directory of the Universal Broker.

Table 9.1, below, identifies all of the options that comprise the UDM Server for UNIX component definition.

Each **Option Name** is a link to detailed information about that component definition option.

Option Name	Description
AUTOMATICALLY_START	Specification for whether or not UDM Server starts automatically when Universal Broker is started.
COMPONENT_NAME	Name by which the clients know the UDM Server.
CONFIGURATION_FILE	Full path name of the UDM Server configuration file.
RUNNING_MAXIMUM	Maximum number of UDM Servers that can run simultaneously.
START_COMMAND	Full path name of the UDM Server program.
WORKING_DIRECTORY	Full path name of the UDM Server working directory.

Table 9.1 UDM Server for UNIX - Component Definition Options

9.3 Configuration

Universal Data Mover Server configuration consists of defining runtime and default values. This section describes the UDM Server configuration options.

9.3.1 Configuration File

The configuration file provides the simplest method of specifying configuration values that will not change with each command invocation.

The Universal Data Mover Server configuration file name (and directory) is specified in the Universal Data Mover Server component definition (see Chapter 12 Universal Data Mover Component Definition Options). The default configuration file name is **`udms.conf`**.

This file can be edited manually with any text editor.

9.3.2 Configuration Options

Table 9.2 identifies all UDM Server for UNIX configuration options.

Each **Option Name** is a link to detailed information about that configuration option.

Option	Description
ACTIVITY_MONITORING	Specification for whether or not product activity monitoring events are generated.
CODE_PAGE	Character code page used to translate text data.
DATA_COMPRESSION	Specification for whether or not data is compressed on all standard I/O files.
DATA_SSL_CIPHER_LIST	SSL cipher suites to use for data session between UDM primary and secondary servers.
EVENT_GENERATION	Events to be generated as persistent events.
FRAME_INTERVAL	Number of UDM transfer blocks transferred before a frame-sync message is sent with network fault tolerance turned on.
INSTALLATION_DIRECTORY	Directory on which UDM is installed.
MESSAGE_LEVEL	Level of messages that UDM will write to the Universal message Catalog (UMC) file.
NETWORK_DELAY	Expected network latency (in seconds).
NLS_DIRECTORY	Directory where the UDM Manager message catalog and code page tables are located.
OUTBOUND_IP	Host or IP address that UDM binds to when initiating outgoing connections to another UDM server.
RECONNECT_RETRY_COUNT	Number of attempts that the UDM Manager will make to re-establish a transfer session when a network fault occurs.
RECONNECT_RETRY_INTERVAL	Number of seconds that UDM will wait between each successive attempt to re-establish a transfer session when a network fault occurs.
RECV_BUFFER_SIZE	Size of the TCP receive buffer for UDM.
SEND_BUFFER_SIZE	Size of the TCP send buffer for UDM.
TCP_NO_DELAY	Specification for whether or not to use TCP packet coalescing.
TMP_DIRECTORY	Directory that UDM Server uses for temporary files.
TRACE_DIRECTORY	Directory that UDM Server uses for its Trace files.
TRACE_FILE_LINES	Maximum number of lines to write to the trace file.
TRACE_TABLE	Size of a wrap-around trace table maintained in memory.
UMASK	File mode creation mask.
USER_SECURITY	User security option.

Table 9.2 UDM Server for UNIX - Configuration Options

9.4 Universal Access Control List

UDM Server uses the Universal Access Control List (UACL) file as an extra layer of security. The UACL file contains UDM Server entries that contain Access Control List (ACL) rules that permit or deny access to the UDM Server.

9.4.1 UACL Entries

The syntax of a UACL entry file is the same as the UDM configuration file.

Table 9.3 identifies all UDM Server for UNIX UACL entries.

Each **UACL Entry Name** is a link to detailed information about that UACL entry.

UACL Entry Name	Description
UDM_ACCESS	Allows or denies access to Universal Data Mover Server services
UDM_MGR_ACCESS	Allows or denies access based on the host name and/or user of the Manager trying to initiate a UDM session

Table 9.3 UDM Server for UNIX - UACL Entries

CHAPTER 10

Universal Data Mover Server for IBM i

10.1 Overview

This chapter provides the following information on the Universal Data Mover (UDM) Server, specific to the IBM i operating system:

- Component Definition
- Configuration
- Universal Access Control List

10.2 Component Definition

All Stonebranch Solutions components managed by Universal Broker have a component definition. The component definition is a text file of options containing component-specific information required by Universal Broker. (For details on how Universal Broker manages components, see the Universal Broker Reference Guide.)

The syntax of a component definition file is the same as a configuration file.

The UDM Server for IBM i component definition is located in the component definition directory of the Universal Broker.

Table 10.1, below, identifies all of the options that comprise the UDM Server for IBM i component definition.

Each **Option Name** is a link to detailed information about that component definition option.

Option Name	Description
AUTOMATICALLY_START	Specification for whether or not UDM Server starts automatically when Universal Broker is started.
COMPONENT_NAME	Name by which the clients know the UDM Server.
CONFIGURATION_FILE	Full path name of the UDM Server configuration file.
RUNNING_MAXIMUM	Maximum number of UDM Servers that can run simultaneously.
START_COMMAND	Full path name of the UDM Server program.
WORKING_DIRECTORY	Full path name of the UDM Server working directory.

Table 10.1 UDM Server for IBM i - Component Definition Options

10.3 Configuration

UDM Server configuration consists of defining runtime and default values. This section describes the Server configuration options.

10.3.1 Configuration File

The configuration file provides the simplest method of specifying configuration values that will not change with each command invocation.

The UDM Server configuration file name is specified in the UDM Server component definition (see Section 10.2 Component Definition). The default name is **`UNVPRD420/UNVCONF(UDMS)`**.

This file can be edited manually with any text editor.

10.3.2 Configuration Options

Table 10.2 identifies all Universal Data Mover Server for IBM i configuration options.

Each **Option Name** is a link to detailed information about that configuration option.

Option	Description
ACTIVITY_MONITORING	Specification for whether or not product activity monitoring events are generated.
CODE_PAGE	Character code page used to translate text data.
CODEPAGE_TO_CCSID_MAP	Specification to use either the internal or external table for code page to CCSID mapping.
DATA_COMPRESSION	Specification for whether or not data is compressed on all standard I/O files.
DATA_SSL_CIPHER_LIST	SSL cipher suites to use for data session between UDM primary and secondary servers.
EVENT_GENERATION	Events to be generated as persistent events.
FRAME_INTERVAL	Number of UDM transfer blocks transferred before a frame-sync message is sent with network fault tolerance turned on.
MESSAGE_LEVEL	Level of messages that UDM will write to the Universal message Catalog (UMC) file.
NETWORK_DELAY	Expected network latency (in seconds).
OUTBOUND_IP	Host or IP address that UDM binds to when initiating outgoing connections to another UDM server.
RECONNECT_RETRY_COUNT	Number of attempts that the UDM Manager will make to re-establish a transfer session when a network fault occurs.
RECONNECT_RETRY_INTERVAL	Number of seconds that UDM will wait between each successive attempt to re-establish a transfer session when a network fault occurs.
RECV_BUFFER_SIZE	Size of the TCP receive buffer for UDM.
SEND_BUFFER_SIZE	Size of the TCP send buffer for UDM.
SIZE_ATTRIB	Default size for file creation of physical files for both data and source file types.
TCP_NO_DELAY	Specification for whether or not to use TCP packet coalescing.
TMP_DIRECTORY	Directory that UDM Server uses for temporary files.
TRACE_FILE_LINES	Maximum number of lines to write to the trace file.
TRACE_TABLE	Size of a wrap-around trace table maintained in memory.
UMASK	File mode creation mask.
USER_SECURITY	User security option.

Table 10.2 UDM Server for IBM i - Configuration Options

10.4 Universal Access Control List

UDM Server uses the Universal Access Control List (UACL) file as an extra layer of security. The UACL file contains UDM Server entries that contain Access Control List (ACL) rules that permit or deny access to the UDM Server.

10.4.1 UACL Entries

The syntax of a UACL entry file is the same as the UDM configuration file.

Table 10.3 identifies all UDM Server for IBM i UACL entries.

Each **UACL Entry Name** is a link to detailed information about that UACL entry.

UACL Entry Name	Description
UDM_ACCESS	Allows or denies access to Universal Data Mover Server services
UDM_MGR_ACCESS	Allows or denies access based on the host name and/or user of the Manager trying to initiate a UDM session

Table 10.3 UDM Server for IBM i - UACL Entries

CHAPTER 11

Universal Data Mover Server Configuration Options

11.1 Overview

This chapter provides detailed information on the configuration options available for use with the Universal Data Mover Server.

The options are listed alphabetically, without regard to any specific operating system.

Section 11.2 Configuration Options Information provides a guideline for understanding the information presented or each option.

11.2 Configuration Options Information

For each configuration option, this chapter provides the following information.

Description

Describes the configuration option and how it is used.

Usage

Provides a table of the following information:

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	<Format / Value>					

Method

Identifies the method used to specify Universal Data Mover Server configuration options:

- Configuration File Keyword

Syntax

Identifies the syntax of the method used to specify the option:

• Format	Specific characters that identify the option.
• Value	Type of value(s) to be supplied for this method.

(Operating System)

Identifies (with a ✓) the operating systems for which each method of specifying the option is valid:

- IBM i
- NonStop (HP NonStop)
- UNIX
- Windows
- z/OS

Values

Identifies all possible values for the specified value type.

Defaults are identified in **[bracketed bold type]**.

<Additional Information>

Identifies any additional information specific to the option.

11.3 Configuration Options List

Table 11.1 identifies all UDM Server configuration options.

Option	Description	Page
ACTIVITY_MONITORING	Specification for whether or not product activity monitoring events are generated.	184
ALLOC_ABNORMAL_DISP	Disposition of data set when an abnormal ending occurs.	183
ALLOC_BLKSIZE	Block size used for newly allocated data sets.	186
ALLOC_DATACLAS	SMS data class used for newly allocated data sets.	187
ALLOC_DIR_BLOCKS	Number of directory blocks for newly allocated partitioned data sets.	188
ALLOC_DSORG	Data set organization used for newly allocated data sets.	189
ALLOC_INPUT_STATUS	Status of data sets being allocated for input.	190
ALLOC_LRECL	Logical record length used for newly allocated data sets.	191
ALLOC_MGMTCLAS	SMS management class used for newly allocated data sets.	192
ALLOC_NORMAL_DISP	Disposition of data set when normal ending occurs.	193
ALLOC_OUTPUT_STATUS	Status of existing data sets being allocated for output.	194
ALLOC_PRIM_SPACE	Primary space allocation used for newly allocated data sets.	195
ALLOC_RECFM	Record format used for newly allocated data sets.	196
ALLOC_SEC_SPACE	Secondary space allocation used for newly allocated data sets.	197
ALLOC_SPACE_UNIT	Space unit in which space is allocated for newly allocated data sets.	198
ALLOC_STORCLAS	SMS storage class used for newly allocated data sets.	199
ALLOC_UNIT	Unit used for newly allocated data sets.	200
ALLOC_VOLSER	Volume serial number used for newly allocated data sets.	201
CODE_PAGE	Character code page used to translate text data.	202
CODEPAGE_TO_CCSID_MAP	Specification to use the internal or external table for code page to CCSID mapping.	204
DATA_COMPRESSION	Specification for whether or not data is compressed on all standard I/O files.	206
DATA_SSL_CIPHER_LIST	SSL cipher suites to use for data session between UDM primary and secondary servers.	207
EVENT_GENERATION	Events to be generated as persistent events.	208
FRAME_INTERVAL	Number of UDM transfer blocks transferred before a frame-sync message is sent with network fault tolerance turned on.	210
INSTALLATION_DIRECTORY	Directory in which UDM Server is installed.	211
LOGON_METHOD	Specification for how users are logged onto the system.	212
MESSAGE_LEVEL	Level of messages that UDM will write.	213
NETWORK_DELAY	Expected network latency (in seconds).	215

Option	Description	Page
NLS_DIRECTORY	Directory where the UDM Server message catalog and code page tables are located.	216
OUTBOUND_IP	Host or IP address that UDM binds to when initiating outgoing connections to another UDM Server.	217
RECONNECT_RETRY_COUNT	Number of attempts that the UDM Manager will make to re-establish a transfer session when a network fault occurs.	218
RECONNECT_RETRY_INTERVAL	Number of seconds that UDM will wait between each successive attempt to re-establish a transfer session when a network fault occurs.	219
RECV_BUFFER_SIZE	Size of the TCP receive buffer for UDM.	220
SEND_BUFFER_SIZE	Size of the TCP send buffer for UDM.	221
SIZE_ATTRIB	Default size for file creation of physical files for both data and source file types.	222
TCP_NO_DELAY	Specification for whether or not to use TCP packet coalescing.	223
TMP_DIRECTORY	Name of directory that UDM Server uses for temporary files.	224
TRACE_DIRECTORY	Name of directory that UDM Server uses for its trace files.	225
TRACE_FILE_LINES	Maximum number of lines to write to the trace file.	226
TRACE_TABLE	Size of a wrap-around trace table maintained in memory.	228
UMASK	File mode creation mask.	229
USER_SECURITY	User security option.	230

Table 11.1 UDM Server Configuration Options

11.4 ACTIVITY_MONITORING

Description

The ACTIVITY_MONITORING option specifies whether or not product activity monitoring events are generated.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	activity_monitoring *option*	√		√	√	√

Values

option is the specification for whether or not product activity monitoring events are generated.

Valid values for *option* are:

- **yes**
  Activate monitoring events.
- **no**
  Deactivate monitoring events.

[Default is no.]

11.5 ALLOC_ABNORMAL_DISP

Description

The ALLOC_ABNORMAL_DISP option is a dynamic allocation option that specifies the disposition of data set when an abnormal ending occurs.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	alloc_abnormal_disp *disposition*					√

Values

disposition is equivalent to the third positional parameter of the JCL DD statement's DISP parameter.

Valid values for *disposition* are:

- **keep**
  Keep the data set.
- **delete**
  Delete the data set.
- **catlg**
  Catalog the data set.
- **uncatlg**
  Un-catalog the data set.

[Default is delete.]

References

Refer to the IBM JCL Reference manual for complete details.

11.6 ALLOC_BLKSIZE

Description

The ALLOC_BLKSIZE option is a dynamic allocation option that specifies the block size used for newly allocated data sets.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	alloc_blksize *size*					√

Values

size is equivalent to the JCL DD statement's BLKSIZE parameter.

Valid values for *size* are any number (size of a block).

[Default is *27998*.]

References

Refer to the IBM JCL Reference manual for complete details.

11.7 ALLOC_DATACLAS

Description

The ALLOC_BLKSIZE option is a dynamic allocation option that specifies the SMS data class used for newly allocated data sets.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	alloc_dataclas *class*					√

Values

class is equivalent to the JCL DD statement's DATACLAS parameter.

Valid values for *class* are any SMS data classes defined in the local environment.

[There is no default.]

References

Refer to the IBM JCL Reference manual for complete details.

11.8 ALLOC_DIR_BLOCKS

Description

The ALLOC_DIR_BLOCKS option is a dynamic allocation option that specifies the number of directory blocks for newly allocated partitioned data sets.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	alloc_dir_blocks *number*					√

Values

number is equivalent to the third positional parameter of the second positional parameter of the JCL DD statement's SPACE parameter.

Valid values for *number* are any number (number of directory blocks to allocate).

[Default is *20*.]

References

Refer to the IBM JCL Reference manual for complete details.

11.9 ALLOC_DSORG

Description

The ALLOC_DSORG option is a dynamic allocation option that specifies the data set organization used for newly allocated data sets.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	alloc_dsorg *organization*					√

Values

organization is equivalent to the JCL DD statement's DSORG parameter.

Valid values for *organization* are:

- **po**
  Partitioned organization
- **ps**
  Physically sequential

[Default is ps.]

References

Refer to the IBM JCL Reference manual for complete details.

11.10 ALLOC_INPUT_STATUS

Description

The ALLOC_INPUT_STATUS option is a dynamic allocation option that specifies the status of data sets being allocated for input.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	alloc_input_status *status*					√

Values

status is equivalent to the first positional parameter of the JCL DD statement's DISP parameter.

Valid values for *status* are:

- **old**
  Allocate the data set exclusively.
- **shr**
  Allocate the data set non-exclusively.

[Default is old.]

References

Refer to the IBM JCL Reference manual for complete details.

11.11 ALLOC_LRECL

Description

The ALLOC_LRECL option is a dynamic allocation option that specifies the logical record length used for newly allocated data sets.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	alloc_lrecl *length*					√

Values

length is equivalent to the first positional parameter of the JCL DD statement's LRECL parameter.

Valid values for *length* are any number (length of the record).

[Default is *1024*.]

References

Refer to the IBM JCL Reference manual for complete details.

11.12 ALLOC_MGMTCLAS

Description

The ALLOC_MGMTCLAS option is a dynamic allocation option that specifies the SMS management class used for newly allocated data sets.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	alloc_mgmtclas *class*					√

Values

class is equivalent to the first positional parameter of the JCL DD statement's MGMTCLAS parameter.

Valid values for *class* are any SMS management classes defined in the local environment.

[There is no default.]

References

Refer to the IBM JCL Reference manual for complete details.

11.13 ALLOC_NORMAL_DISP

Description

The ALLOC_NORMAL_DISP option is a dynamic allocation option that specifies the disposition of data set when normal ending occurs.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	alloc_normal_disp *disposition*					√

Values

disposition is equivalent to the second positional parameter of the JCL DD statement's DISP parameter.

Valid values for *disposition* are:

- **keep**
  Keep *the data set.*
- **delete**
  Delete the data set.
- **catlg**
  Catalog the data set.
- **uncatlg**
  Un-catalog the data set.

[Default is catlg.]

References

Refer to the IBM JCL Reference manual for complete details.

11.14 ALLOC_OUTPUT_STATUS

Description

The ALLOC_OUTPUT_STATUS option is a dynamic allocation option that specifies the status of existing data sets being allocated for output.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	alloc_output_status *status*					√

Values

status is equivalent to the first positional parameter of the JCL DD statement's DISP parameter.

Valid values for *status* are:

- **new**
  Create new data set.
- **shr**
  Allocate the data set non-exclusively.
- **old**
  Allocate the data set exclusively.
- **mod**
  Either create a new data set, for exclusive use, or allocate a sequential data set exclusively and add records to the end of it.

[Default is old.]

References

Refer to the IBM JCL Reference manual for complete details.

11.15 ALLOC_PRIM_SPACE

Description

The ALLOC_PRIM_SPACE option is a dynamic allocation option that specifies the primary space allocation used for newly allocated data sets.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	alloc_prim_space *space*					√

Values

space is equivalent to the first sub-parameter of the second sub-parameter of the JCL DD statement's SPACE parameter.

Valid values for space are any number (number of space units to allocate).

[Default is 15.]

References

Refer to the IBM JCL Reference manual for complete details.

11.16 ALLOC_RECFM

Description

The ALLOC_RECFM option is a dynamic allocation option that specifies the record format used for newly allocated data sets.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	alloc_recfm *format*					√

Values

format is equivalent to the JCL DD statement's RECFM parameter.

Valid values for *format* are dependent on the data set organization and access method used. The following values are valid for both partitioned and sequential data sets:

- **F**[**B**][**A**|**M**]
  Fixed, optionally blocked, and optionally either ANSI or Machine control characters.
- **V**[**B**][**A**|**M**|**S**]
  Variable, optionally blocked, and optionally either ANSI or Machine control characters, or spanned.

[Default is VB.]

References

Refer to the IBM JCL Reference manual for complete details.

11.17 ALLOC_SEC_SPACE

Description

The ALLOC_SEC_SPACE option is a dynamic allocation option that specifies the secondary space allocation used for newly allocated data sets.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	alloc_sec_space *space*					√

Values

space is equivalent to the second sub-parameter of the second sub-parameter of the JCL DD statement's SPACE parameter.

Valid values for *space* are any number (number of space units to allocate).

[Default is 15.]

References

Refer to the IBM JCL Reference manual for complete details.

11.18 ALLOC_SPACE_UNIT

Description

The ALLOC_SPACE_UNIT option is a dynamic allocation option that specifies the space unit in which space is allocated for newly allocated data sets.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	alloc_space_unit *space*					√

Values

space is equivalent to the first sub-parameter of the JCL DD statement's SPACE parameter.

Valid values for *space* are:

- *number*
  Block length or record length
- **cyl**
  Cylinder allocation
- **trk**
  Track allocation

[Default is trk.]

References

Refer to the IBM JCL Reference manual for complete details.

11.19 ALLOC_STORCLAS

Description

The ALLOC_STORCLAS option is a dynamic allocation option that specifies the SMS storage class used for newly allocated data sets.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	alloc_storclas *class*					√

Values

class is equivalent to the JCL DD statement's STORCLAS parameter.

Valid values for *class* are any SMS storage classes defined in the local environment.

[There is no default.]

References

Refer to the IBM JCL Reference manual for complete details.

11.20 ALLOC_UNIT

Description

The ALLOC_UNIT option is a dynamic allocation option that specifies the unit used for newly allocated data sets.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	alloc_unit *unit*					√

Values

unit is equivalent to the JCL DD statement's UNIT parameter.

Valid values for *unit* are:

- *number*
  Device number
- *name*
  Unit generic or group name

[Default is SYSALLDA.]

References

Refer to the IBM JCL Reference manual for complete details.

11.21 ALLOC_VOLSER

Description

The ALLOC_VOLSER option is a dynamic allocation option that specifies the volume serial number used for newly allocated data sets.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	alloc_volser *number*					√

Values

number is equivalent to the sub-parameter SER of the JCL DD statement's VOL parameter.

Valid values for *number* are any Volume serial number.

[There is no default.]

References

Refer to the IBM JCL Reference manual for complete details.

11.22 CODE_PAGE

Description

The CODE_PAGE option specifies the character code page that is used to translate text data received and transmitted over the network.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	codepage *codepage*	√		√	√	√

Values

codepage can be any character code page.

Note: UTF-8 is not a supported codepage value for the CODE_PAGE option. UTF-8 codepage is valid only for text file data translation. Consequently, it can be specified only on the UDM open script statement.

(Table 21.13 Character Code Pages identifies the character code pages provided for UDM by Stonebranch Inc.)

Default

The default code page is different for different operating systems:

IBM i

- IBM037

UNIX

• ISO8859-1 (8-bit ASCII)	ASCII-based operating systems
• IBM1047 (EBCDIC)	Non-IBM i, EBCDIC-based operating system

z/OS

• ISO8859-1 (8-bit ASCII)	ASCII-based operating systems
• IBM1047 (EBCDIC)	Non-IBM i, EBCDIC-based operating system

UTT Files

Universal Translate Table (UTT) files are used to translate between Unicode and the local single-byte code page.

Platform	Location
IBM i	UNVNLS file in the Stonebranch Solutions installation library (default is UNVPRD420). The value *codepage* is the base file name of the UTT file. UTT files are stored as members without the UTT extension.
z/OS	Members of the PDS allocated to the Broker ddname UNVNLS. The value *codepage* is a member name in the library.
UNIX	Directory specified by NLS_DIRECTORY (default is **`/opt/universal/nls`**). The value *codepage* is the base file name of the UTT file. All UTT files end with an extension of **`.utt`**.

Table 11.2 UTT File Locations

11.23 CODEPAGE_TO_CCSID_MAP

Description

CAUTION: This option is intended only for use by IBM i specialists who fully understand code pages, CCSIDs, how the two relate to each other, and how IBM i uses CCSIDs for data translation between data streams and files. If a code page and CCSID are not correctly matched, data corruption will occur.

The CODEPAGE_TO_CCSID_MAP option specifies whether to use the internal table or external table for code page to CCSID mapping.

An internal table provides code page to CCSID mapping for the code page specified by the open command. The mapping only occurs if the CCSID is required for text file mapping. For the LIB file system, this includes mapping text to source physical files or to data files with an associated DDS file. All files in the root and QOpenSys file systems have associated CCSIDs.

This CCSID is not the same as the CCSID attribute associated with the attrib command; the UDM CCSID attribute determines the CCSID of the target file if the file does not exist. This CCSID in the mapping table is used as the CCSID associated with the attrib command when the default value, CODEPAGE, is specified; it also identifies the data stream CCSID, allowing the operating system to translate the code page translated data stream to the file. However, data written to or read from a file, record, or field with a CCSID of 65535 (or 'HEX') will not be translated.

The code page (and the mapped CCSID) from the open command is used for data mapping between the two parties involved in a data transfer. For IBM i, the code page also is used for mapping the data stream to or from the IBM i file, whether a LIB or HFS transfer.

Under normal circumstances, the external mapping table will not be needed. This external table replaces the internal table, so all potentially needed code page to CCSID mappings must be provided in the external table.

Stonebranch Solutions for IBM i provides an example external table in file **CP2CCSID_X**, in product library **UNVPRD420**. For UDM to use the external table, create a single member file with the name **CDPG2CCSID** in the installation library. The example file may be used as a template by copying it to **CDPG2CCSID** in the same library.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	codepage_to_ccsid_map_opt *table*	✓				

Value

table is the specification for which table to use:

- **error**
  Use the external table; if it is not found, report an error.
- **quiet**
  Use the external table; if it is not found, use the internal table. No message is issued.
- **internal**
  Use the internal table.

[Default is internal.]

11.24 DATA_COMPRESSION

Description

The DATA_COMPRESSION option specifies whether or not data standard I/O file transmissions across the network should be compressed. Optionally, it also can specify the compression method to use.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	compress *option*[,*method*]	√		√	√	√

Values

option is either of the following values:

- **yes**
  Data compression is required. All data in standard I/O file transmissions is compressed regardless of the UDM Manager DATA_COMPRESSION option value.
- **no**
  Data compression is not required. However, data compression still can be requested via the UDM Manager DATA_COMPRESSION option.

[Default is no.]

method is either of the following values:

- **zlib**
  Data is compressed using ZLIB compression algorithm. This method usually results in a very high compression rate, but tends to be somewhat CPU-intensive. It is recommended in environments where controlling a process's CPU usage is not necessarily a priority.
- **hasp**
  Data is compressed using the HASP compression algorithm. This method is less CPU-intensive than the ZLIB method. It is recommended in environments where controlling CPU usage is a priority. With HASP, the compression rate, while still very good, tends to be a little less than what is possible with the ZLIB.

[Default is zlib.]

11.25 DATA_SSL_CIPHER_LIST

Description

The DATA_SSL_CIPHER_LIST option specifies the acceptable and preferred SSL cipher suites to use for the data session on which file data is transferred between UDM primary and secondary servers.

The SSL protocol uses the cipher suites to specify which encryption and message authentication (or message digest) algorithms to use.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	data_ssl_cipher_list *list*	√		√	√	√

Values

list is a comma-separated list of SSL cipher suites. The cipher suites should be listed with the most preferred cipher suite first and the least preferred cipher suite last.

Table 21.15 SSL Cipher Suites for UDM identifies the list of SSL cipher suites provided for UDM by Stonebranch Inc.

Note: In order to establish a transfer session without using SSL for the data session, the **`NULL-NULL`** cipher must be specified in the cipher list for any UDM Server involved in the session and in the encrypt option of the open command.

Default

The default list of SSL cipher suites is:

**RC4-SHA,RC4-MD5,AES256-SHA,AES128-SHA,DES-CBC3-SHA,
DES-CBC-SHA,NULL-SHA,NULL-MD5**

11.26 EVENT_GENERATION

Description

The EVENT_GENERATION option specifies which events are to be generated and processed as persistent events.

A persistent event record is saved in a Universal Enterprise Controller (UEC) database for long-term storage.

(For a list of all event types for all Stonebranch Solutions components, see the Universal Event Subsystem 4.2.0 Event Definitions document.)

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	event_generation *types*	√		√	√	√

Values

type specifies a comma-separated list of event types. It allows for all or a subset of all potential event message types to be selected.

Event type ranges can be specified by separating the lower and upper range values with a dash (**-**) character.

Event types can be selected for inclusion or exclusion:

- Inclusion operator is an asterisk (*****).
- Exclusion operator is **X** or **x**.

Examples

- 100,101,102
  Generate event types 100, 101, and 102.
- 100-102
  Generate event types 100 through 102.
- 100-102,200
  Generate event types 100 through 102 and 200.
- *
  Generate all event types.
- *,X100
  Generate all event types except for 100.
- x*
  Generate no event types.
- *,X200-250,X300
  Generate all event types except for 200 through 250 and 300.

[Default is *X (no event types).]**

11.27 FRAME_INTERVAL

Description

The FRAME_INTERVAL options sets the number of UDM transfer blocks transferred before a frame-sync message is sent when UDM is operating with network fault tolerance on (see Section 6.44 NETWORK_FAULT_TOLERANT).

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	frame_interval *number*	√		√	√	√

Values

number can be any number.

[Default is *100*.]

Note: This value should not be changed without direction from Stonebranch, Inc. Customer Support. Changing this value could degrade UDM performance.

11.28 INSTALLATION_DIRECTORY

Description

The INSTALLATION_DIRECTORY option specifies the directory in which Universal Data Mover is installed.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	installation_directory *directory*			√	√	

Values

directory is any directory.

Defaults

UNIX

[Default is `/opt/universal/udmsrv`.]

Windows

[Default is the UDM Server installation file: `c:\Program Files\Universal\udmsrv`.]

11.29 LOGON_METHOD

Description

The LOGON_METHOD option specifies the user's log on method.

If the UCMD Server is configured for user security (see the USER_SECURITY option), the log on method determines how the user is logged onto the Windows system.

If security is inactive, LOGON_METHOD is ignored.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	logon *option*				√	
Manager Override	n/a					

Values

option is the user's log on method.

Valid values for *option* are:

- **batch**
  Windows log on type is `batch`. A batch log on prevents the user account from interacting with the desktop. The user ID logging on as a batch user requires the Windows User Right "Log on as a batch job." If the user does not have this right, the user authentication will fail.
- **interactive**
  Windows log on type is `interactive`. An interactive log on permits the user account to interact with the desktop. A user account must have the "Allow log on locally" privilege granted to it to successfully authenticate with an interactive logon.

[Default is interactive.]

11.30 MESSAGE_LEVEL

Description

The MESSAGE_LEVEL option specifies the level of messages to write.

It also specifies, optionally, whether or not to write a date and time stamp with each message.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	message_level *level*[,*time*]	√		√	√	√

Values

level indicates either of the following level of messages:

- **trace**
  Activates tracing and generates a trace file to which UDM writes trace messages used for debugging.

  Note: Use **trace** only as directed by Stonebranch, Inc. Customer Support.
- **audit**
  Writes audit, informational, warning, and error messages.
- **info**
  Writes informational, warning, and error messages.
- **warn**
  Writes warning and error messages.
- **error**
  Writes error messages only.

[Default is info.]

time specifies either of the following:

- **time**
  Include a time and date stamp on each message.
- **notime**
  Do not include a time and date stamp on each message.

[Default is time.]

Trace Files

UNIX

Trace file name is **`udmsrv-N-S.trc`** where:

- N = Component ID assigned to this instance of the Server,
- S = Sequence number.

The trace file is created in the trace directory, as specified by the TRACE_DIRECTORY option (default is **`/var/opt/universal/trace`**).

z/OS

There are two possible destinations of the trace data:

1. If ddname **`UNVTRMDL`** is defined in the UBROKER started task procedure, a sequential data set is created using the data set allocated to **`UNVTRMDL`** as a model. The dynamically allocated trace data set name is **`#HLQ.UDM.Dyymmdd.Thhmmss.Cnnnnnnn`** where:
   - **`#HLQ`** is the data set name allocated on the **`UNVTRMDL`** ddname
   - **`yymmdd`** is the year, month, day
   - **`hhmmss`** is the hour, minute, second the data set was allocated
   - **`nnnnnnn`** is the last seven digits of the Server's component ID in hexadecimal format
2. If ddname **`UNVTRMDL`** is not defined in the UBROKER started task procedure, member name **Dnnnnnns** is created in the PDS or PDS/E allocated to the **`UNVTRACE`** ddname, where, **nnnnnn** is the last six digits of the Server's component ID in hexadecimal format, and **s** is the component ID's sequence number (from `0` - `F`). Each time a server is restarted, its sequence number is incremented. If a server is restarted more than 15 times, tracing is disabled.

Depending on the error condition being diagnosed, it is possible that the member name of the **`UNVTRACE`** PDS or PDS/E is not created. If this occurs, the **`UNVTRMDL`** ddname must be used to create a sequential data set name.

The records written to PDS and PDS/E members cannot be wrapped, so the TRACE_FILE_LINES limit has no effect on the maximum number of trace records written to the member.

IBM i

Trace file name is **`*CURLIB/UNVTRCUDMS(Txxxyyyyyy)`** where:

- **`xxx`** = first three digits of the process ID, in hexadecimal, assigned to this instance of the Server by IBM i.
- **`yyyyyy`** = last six digits of the component ID, in hexadecimal, assigned to this instance of the Server by the Broker.

3. These values are used to generate a unique identifier for each invocation of the UDMSRV program.
4. The default library for TRACE is the current library (**`*CURLIB`**) of the UBROKER process, not the user process. The UBROKER current library is the temporary library designated during the Stonebranch Solutions installation process; by default, the UBROKER current library is **`UNVTMP420`**.

11.31 NETWORK_DELAY

Description

The NETWORK_DELAY option sets the expected network latency (in seconds).

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	network_delay *number*	√		√	√	√

Values

number is any number.

[Default is *120*.]

11.32 NLS_DIRECTORY

Description

The NLS_DIRECTORY option specifies the directory name where the UDM Server can find its message catalog and code page tables.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	nls_directory *directory*			√	√	

Values

directory can be any directory.

Full path names are recommended.

Relative path names are relative to the **`universal`** installation directory.

UNIX

[Default is **`/opt/universal/nls`**.]

Windows

[Default is **`..\nls`**.]

11.33 OUTBOUND_IP

Description

The OUTBOUND_IP option sets the host or IP address that UDM binds to when initiating outgoing connections to another UDM server.

By default, this configuration option is not set.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	outbound_ip *host*	√		√	√	√

Values

host is the required IP address.

[There is no default.]

11.34 RECONNECT_RETRY_COUNT

Description

The RECONNECT_RETRY_COUNT option sets the number of attempts that the primary transfer server will make in order to re-establish a transfer session with the secondary transfer server in a three-party transfer session when a network fault occurs.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	reconnect_retry_count *number*	√		√	√	√

Values

number is any number.

[Default is *20*.]

11.35 RECONNECT_RETRY_INTERVAL

Description

The RECONNECT_RETRY_INTERVAL option sets the number of seconds that UDM will wait between each successive attempt to reestablish a transfer session when a network fault occurs.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	reconnect_retry_interval *number*	√		√	√	√

Values

number is any number (of seconds).

[Default is *60*.]

11.36 RECV_BUFFER_SIZE

Description

The RECV_BUFFER_SIZE option sets the size (in bytes) of the TCP receive buffer for UDM.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	recv_buffer_size *size*	√		√	√	√

Values

size is the number of bytes.

[Default is *0*.]

(The default, **0,** instructs UDM to use the operating system default.)

Note: The size of the TCP receive buffer should be changed only when performance tweaking is necessary. Changing this value could affect performance adversely.

11.37 SEND_BUFFER_SIZE

Description

The SEND_BUFFER_SIZE option sets the size (in bytes) of the TCP send buffer for UDM.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	send_buffer_size *size*	√		√	√	√

Values

size is the number of bytes.

[Default is *0*.]

(The default, **0,** instructs UDM to use the operating system default.)

Note: The size of the TCP send buffer should be changed only when performance tweaking is necessary. Changing this value could affect performance adversely.

11.38 SIZE_ATTRIB

Description

The SIZE_ATTRIB option sets the default size (number of records) for file creation of physical files for both data and source file types.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	size_attrib *size*	√				

Values

size is the number of records.

Other than the leading * for *NOMAX, any valid value can be entered for file size.

The values for Initial Number of Records, Increment Number of Records, and Maximum increments are comma-separated, not space-separated.

Examples:

- 1000,100,100
- 10000,499
- 50000

To specify *NOMAX, enter 'NOMAX'.

UDM uses this value as the default file SIZE attribute. (See Table 18.2 IBM i-Specific LIB File Attributes for Creating New Files for details.)

[Default is *empty string*.]

11.39 TCP_NO_DELAY

Description

The TCP_NO_DELAY option specifies whether or not to use TCP packet coalescing.

The packet coalescing algorithm, which can delay the sending of small amounts of data over the network, is designed to improve network congestion. However, since it can have a significantly negative effect on the performance of UDM, TCP_NO_DELAY specifies – by default – not to use TCP packet coalescing.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	tcp_no_delay *option*	√		√	√	√

Values

option is the specification for whether or not to use TCP packet coalescing.

Valid values for *option* are:

- **yes**
  Do not use TCP packet coalescing.
- **no**
  Use TCP packet coalescing.

[Default is yes.]

11.40 TMP_DIRECTORY

Description

The TMP_DIRECTORY option specifies the name of the directory that the UDM Server uses for temporary files.

IBM i

Do not include this directory in any system or backup that requires an exclusive lock on the directory while UDM is running.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	tmp_directory *directory*	√		√	√	√

Values

directory is the name of the directory.

It should specify a fully qualified path name.

Defaults

UNIX

Default is ***/var/opt/universal/tmp*****.**

Windows

[Default is ***..\tmp*****.**

z/OS

Default is ***/tmp*****.**

IBM i

Default is ***/tmp*****.**

11.41 TRACE_DIRECTORY

Description

The TRACE_DIRECTORY option specifies the directory name that the UDM Server uses for its trace files.

Relative path names are relative to the UDM Server installation directory. Full path names are recommended.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	trace_directory *directory*			√	√	

Values

directory is the name of the directory.

UNIX

[Default is `/var/opt/universal/trace`.]

Windows

[Default is `c:\program\files\universal\udmsrv`.]

11.42 TRACE_FILE_LINES

Description

The TRACE_FILE_LINES option specifies the maximum number of lines to write to the trace file.

When the maximum number of lines has been reached, the trace file will wrap around and start writing trace entries after the trace header lines.

Tracing is activated, and a trace file is generated, when the MESSAGE_LEVEL option is set to TRACE.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	trace_file_lines *number*	√		√	√	√

Values

z/OS

The average size of a trace file line is 50 characters.

The trace file wrapping is only supported with sequential data sets that have a fixed record format. Partitioned data sets or variable record formats are not supported.

[Default is *200,000*.]

UNIX

The average size of a trace file line is 50 characters.

The default is a very large value of `200,000`. If space is limited in the trace file directory (specified with the TRACE_DIRECTORY option), set this to a smaller value.

IBM i

The current record length is 384 characters.

As allocated on the AS/400 (i5), the maximum number of records is 509,000. Increasing TRACE_FILE_LINES beyond this value requires manual adjustment of the file size via the CHGPF command.

Note:

1. The file is not created until the first time Universal Data Mover Server trace is used.
2. Deleting the trace file will reset the maximum number of records to 509,000.
3. To clear the trace file without resetting the maximum number of records the file may contain, use the RMVM FILE(UNVTMP420/UNVTRCUDM) MBR(*ALL) command. Substitute the default library designated during product installation if different from UNVTMP420.
4. WARNING: Setting this number of records larger than the maximum number of records allocated to the trace file will result in a prompt sent to QSYSOPR. Unless a default response is set for this message (CPA5305), the PROCESS WILL HANG waiting for the QSYSOPR response.

[Default is *200,000*.]

If space is limited in the ASP where the trace file is located, set this to a smaller value. To avoid trace file wrapping, set this to a larger value. Please read the **`trace_file_lines`** information in the UNVCONF(UDMS) file member before increasing this value.

11.43 TRACE_TABLE

Description

The TRACE_TABLE option specifies the size of a wrap-around trace table maintained in memory.

The trace table is written to a file / data set when the program ends under the conditions specified by value *condition*. Tracing is activated, and a trace file is generated, when the MESSAGE_LEVEL option is set to **trace**.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	trace_table *size,condition*	√		√	√	√

Values

size is the size (in bytes) of the table.

The size can be suffixed with either of the following characters:

- **M** indicates that the size is specified in megabytes
- **K** indicates that the size is specified in kilobytes

For example, **50M** indicates that 50 X 1,048,576 bytes of memory is allocated for the trace table.

Note: A value of **0** indicates that the trace table is not used.

condition is the condition under which the trace table is printed.

Possible values for *condition* are:

- **error**
  Write the trace table if the program ends with a non-zero exit code.
- **always**
  Write the trace table when the program ends regardless of the exit code.
- **never**
  Never write the trace table.

11.44 UMASK

Description

The UMASK option specifies the file mode creation mask. It affects the file permission bits of newly created files.

All files are created with a permission mode of 666, which is read-write permission for the owner, group, and other permission categories. UDM uses UMASK to turn off selected permission bits by subtracting the value of UMASK from mode 666.

Note: UMASK is supported only for the Hierarchical File System (HFS).

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	umask *number*	√		√		√

Values

number can be any number, 001 to 666.

[Default is *026*.]

The default value (**026**) results in file permission 640 (666 - 026 = 640), which is:

- read-write for the owner
- read for the group
- none for others

References

Refer to the UNIX man page umask(1) for complete details.

11.45 USER_SECURITY

Description

The USER_SECURITY option specifies the user security option.

If user security is activated, the UDM Server logs the user onto the system, and the command is run with the user's identity. If user security is not activated, the command runs with the same identity as the Server.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Configuration File Keyword	security *option*	√		√	√	√

Values

option is either of the following values:

z/OS

- **default**
  Use z/OS SAF user authentication. User ID must have OMVS segment.
- **none**
  No user security

UNIX

- **default**
  Use UNIX default user authentication method, `/etc/passwd`. **[default]**
- **none**
  No user security.
- **pam**
  Use the pluggable Authentication Modules (PAM) interface.
- **trusted**
  User HP Trust Security authentication.

IBM i

- **default**
  Use IBM i security.
- **none**
  No user security

CHAPTER 12

Universal Data Mover Component Definition Options

12.1 Overview

This chapter provides detailed information about the options that comprise Universal Data Mover (UDM) component definitions.

The options are listed alphabetically, without regard to any specific operating system.

Section 12.2 Component Definition Information provides a guideline for understanding the information presented for each component definition option.

12.2 Component Definition Information

For each component definition option, this chapter provides the following information.

Description

Describes the option and how it is used.

Usage

Provides a table of the following information:

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Component Definition Keyword	<Format / Value>					

Method

Identifies the method used for specifying a Universal Data Mover component definition option:

- Component Definition Keyword

Syntax

Identifies the syntax of the method used to specify the option:

• Format	Specific characters that identify the option.
• Value	Type of value(s) to be supplied for this method.

(Operating System)

Identifies (with a ✓) the operating systems for which the method of specifying the option is valid:

- IBM i
- NonStop (HP NonStop)
- UNIX
- Windows
- z/OS

Values

Identifies all possible values for the specified value type.

Defaults are identified in **[bracketed bold type]**.

12.3 Component Definition Options List

Table 12.1 identifies all of the options that can comprise a Universal Data Mover component definition.

Component	Description	Page
AUTOMATICALLY_START	Specification for whether or not UDM Server starts automatically when Universal Broker is started.	235
COMPONENT_NAME	Name by which the clients know the UDM Server.	236
CONFIGURATION_FILE *	Name of the UDM Server configuration file.	237
RUNNING_MAXIMUM	Maximum number of UDM Servers that can run simultaneously.	238
START_COMMAND *	Program name of the UDM Server.	239
WORKING_DIRECTORY *	Directory used as the working directory of the UDM Server.	240
* These options are required in all component definitions.		

Table 12.1 Universal Data Mover Component Definition Options

12.4 AUTOMATICALLY_START

Description

The AUTOMATICALLY_START option specifies whether or not the UDM Server starts automatically when Universal Broker is started.

Note: AUTOMATICALLY_START is optional in a component definition.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Component Definition Keyword	auto_start *option*	√	√	√	√	√

Values

option is the specification for how the UDM Server is started.

The only valid value for *option* is:

- **no**
  UDM Server is not started automatically when Universal Broker is started. It is started only on demand.

12.5 COMPONENT_NAME

Description

The COMPONENT_NAME option specifies the name of the UDM Server.

Component start requests refer to UDM Server by this name.

Note: COMPONENT_NAME is optional in a component definition. If it is not specified, the file name is used as the component name.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Component Definition Keyword	component_name *name*	√	√	√	√	√

Values

name is the name of the UDM Server.

There is only one valid value for *name*: **`udm`**. (This is the name of the UDM Server component definitions file / member.)

Note: This name should not be changed.

12.6 CONFIGURATION_FILE

Description

The CONFIGURATION_FILE option specifies the name of the UDM Server configuration file.

Note: CONFIGURATION_FILE is required in a component definition.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Component Definition Keyword	configuration_file *member* **or** configuration_file *filename*	√	√	√	√	√

Values

member / *filename* is the name of the configuration member / file.

IBM i

Configuration file name can be any valid file name. It can be edited manually with SEU, EDTF, or any other installed source file editor. The default file name is **`UNVPRD420/UNVCONF(UDMS)`**.

HP NonStop

Full path name of the configuration file. The file name can be any valid file name. The installation default is **`$SYSTEM.UNVCONF.UDMSCFG`**.

UNIX

Full path name of the configuration file. The file name can be any valid file name. The installation default is **`/etc/universal/udms.conf`**.

Windows

Full path name of the configuration file. The file name can be any valid file name. The installation default is **`c:\Documents and Settings\All Users\Application Data\Universal\conf\udms.conf`**.

z/OS

Member name of the component configuration file in the **`UNVCONF`** library allocated to the Universal Broker ddname **`UNVCONF`**. The installation default is **`UDSCFG00`**.

12.7 RUNNING_MAXIMUM

Description

The RUNNING_MAXIMUM option specifies the maximum number of UDM Servers that can run simultaneously.

If this maximum number is reached, any command received to start the component is rejected.

Note: RUNNING_MAXIMUM is optional in a component definition.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Component Definition Keyword	running_max *max*	√	√	√	√	√

Values

max is the maximum number of UDM Servers that can run simultaneously.

[Default is *100*.]

12.8 START_COMMAND

Description

The START_COMMAND option specifies the full path name (member name for z/OS) of the Universal Data Mover Server program.

Optionally, START_COMMAND also can specify command line options.

Note: START_COMMAND is required in a component definition.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Component Definition Keyword	start_command *member* **or** start_command *name[options]*	√	√	√	√	√

Values

member / *name* is the program name of UDM Server.

options is the optional list of command line options.

z/OS

name is the program object of the UDM Server. The program object must be in the Universal Broker's search order for loading program objects. The default location is the **SUNVLOAD** library allocated to the Universal Broker's **STEPLIB** ddname.

HP NonStop and UNIX

name is the full path name of the UDM Server program.

Windows

name is the full path name of the UDM Server program.

IBM i

name is the fully qualified name of the UDM Server program. The default is ***LIBL/UDMSRV**.

12.9 WORKING_DIRECTORY

Description

The WORKING_DIRECTORY option specifies the full path name of the directory used as the working directory of UDM Server.

Note: WORKING_DIRECTORY is required in a component definition.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
Component Definition Keyword	working_directory *directory*	√	√	√	√	√

Values

directory is the full path name of the working directory.

[Default is (*.*).

HP NonStop, UNIX, Windows

directory is the full path name of the directory UDM Server uses as its working directory.

z/OS

directory is the HFS directory name that the UDM Server uses as its working directory.

IBM i

working_directory serves as a required placeholder only.

Note: Do not change this directory.

CHAPTER 13

Universal Data Mover UACL Entries

13.1 Overview

This chapter provides detailed information on the Universal Access Control List (UACL) entries available for use with Universal Data Mover.

The UACL entries are listed alphabetically, without regard to any specific operating system.

Section 13.2 UACL Entries Information provides a guideline for understanding the information presented for each UACL entry.

13.2 UACL Entries Information

For each UACL entry, this chapter provides the following information.

Description

Describes the UACL entry and how it is used.

Usage

Provides a table of the following information:

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
UACL File Keyword	<Type / Rule>					

Method

Identifies the method used for specifying a UACL entry:

- UACL FIle Keyword

Syntax

Identifies the syntax of the method used for a UACL entry:

• Type	Stonebranch Solutions component to which the rule applies.
• Rule	Client's identity, request to which the entry pertains, and security attributes that the entry enforces.

(Operating System)

Identifies (with a ✓) the operating systems for which the method of specifying the UACL entry is valid:

- IBM i
- NonStop (HP NonStop)
- UNIX
- Windows
- z/OS

Values

Identifies all possible values for the fields in a UACL entry rule.

Defaults are identified in **[bracketed bold type]**.

13.3 UACL Entries List

Table 13.1 identifies all Universal Data Mover UACL entries.

UACL Entry	Description	Page
UDM_ACCESS	Allows or denies access to Universal Data Mover Server services. There are two forms to this entry: • **`udm_access`** • **`udm_cert_access`**	245
UDM_MGR_ACCESS	Allows or denies access based on the host name and/or user of the Manager trying to initiate a UDM session.	247

Table 13.1 Universal Data Mover UACL Entries

13.4 UDM_ACCESS

Description

A UDM_ACCESS UACL entry either allows or denies access to Universal Data Mover Server services.

If access is permitted, UDM_ACCESS also specifies whether or not user authentication is required.

There are two forms of the UDM_ACCESS entry based on the client identification method:

- **udm_access** form is for IP-based client identification.
- **udm_cert_access** is for X.509 certificate-based client identification.

A **udm_access** UACL entry is matched if all of the following occur:

- Request comes from an IP address identified by *host*.
- Remote end is executing as user *remote_user*.
- Remote user is requesting to execute a command as local user *local_user*.

A **udm_cert_access** UACL entry is matched if both of the following occur:

- Request comes from a client with a certificate identifier of *certid*.
- Remote user is requesting to execute a command as local user *local_user*.

The first matching rule is used to control access.

See Section 7.5.2 UACL Entries in the Infitran 4.2.0 User Guide for details on *host*, *remote_user*, and *local_user* specification syntax.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
UACL File Keyword	udm_access *host,remote_user,local_user,access, auth* udm_cert_access *certid,local_user,access,auth*	√		√	√	√

Values

Valid values for *access* are:

- **deny**
  Service is denied. A message is returned to the remote end. The connection is closed.
- **allow**
  Service is accepted and processed.

Valid values for *auth* are:

- **auth**
  Local user account must be authenticated. The Manager must provide a proper password for the account.
- **noauth**
  Local user account does not require user authentication.

  Note: **noauth** should be used with care. Turning off user authentication may violate your local security policies on the Server system.

13.5 UDM_MGR_ACCESS

Description

A UDM_MGR_ACCESS UACL entry either allows or denies access to Universal Data Mover Server services based on the host name and/or user of the Manager trying to initiate a UDM session.

If access is permitted, UDM_MGR_ACCESS also specifies whether or not user authentication is required.

A **`udm_mgr_access`** UACL entry is matched if all of the following occur:

- Request comes from a Manager initiated on host name identified by host. This is the machine host name, which may or may not be equivalent to the host DNS name.
- Manager is executing as user *manager_user*.
- Manager is requesting to execute a command as local user *local_user*.

The first matching rule is used to control access.

See Section 7.5.2 UACL Entries in the Infitran 4.2.0 User Guide for details on *host*, *manager_user*, and *local_user* specification syntax.

CAUTION: Managers earlier than 3.2.0 supply neither a manager host name nor a manager user ID. Therefore, other than host name **ALL** and Manager user ID *****, requests from managers earlier than 3.2.0 will never match a rule. Setting this rule in networks containing managers earlier than 3.2.0 requires careful planning.

Usage

Method	Syntax	IBM i	NonStop	UNIX	Windows	z/OS
UACL File Keyword	udm_mgr_access *manager_host,manager_user, local_user,access,auth*	√		√	√	√

Values

Valid values for *access* are:

- **deny**
  Service is denied. A message is returned to the remote end. The connection is closed.
- **allow**
  Service is accepted and processed.

Valid values for *auth* are:

- **auth**
  Local user account must be authenticated. The Manager must provide a proper password for the account.
- **noauth**
  Local user account does not require user authentication.

  Note: **noauth** should be used with care. Turning off user authentication may violate your local security policies on the Server system.

CHAPTER 14

UDM Commands

14.1 Overview

This chapter provides detailed information on UDM commands.

14.1.1 UDM Commands List

Table 14.1 identifies all of the UDM commands.

Detailed information on each command is provided on the identified page.

Name	Description	Page
appenddata	Appends a line of text to the end of an existing data element, or creates a new data element containing that line of text.	252
attrib	Sets the file system attributes that govern the transfer operations on the host with the specified logical name.	253
break	Stops iterating through a **`forfiles`** loop and picks up execution at the script line immediately following the end statement marking the end of the **`forfiles`** loop.	258
call	Loads and executes a command script.	259
cd	Changes the working directory (on UNIX, Windows, IBM i, and file system HFS) or if z/OS, the current data set qualifiers (for DSN and DD file systems) on the specified logical machine to the specified path.	261
close	Closes the current transfer session.	262
closelog	Closes the open log file.	263
compare	Compares two strings of data	264
copy	Initiates a copy operation.	265
copydir	Initiates a copy operation that recurses into subdirectories.	267

Name	Description	Page
data	Defines an in-stream data element that can be passed as input for other commands.	269
debug	Turns debug information on and off.	270
delete	Deletes a file (or series of files if file-spec contains any wildcards) from the transfer server with the corresponding logical name.	271
deletestring	Removes a substring from an existing string.	272
echo	Sends text to standard out (stdout).	274
echolog	Sends text to an open log file.	275
exec	Executes system commands on remote machines.	277
execsap	Executes SAP events.	281
exit	Exits the UDM Manager (same as the **`quit`** command).	284
filesys	Sets the file system with which the server with the specified logical name is working.	285
filetype	Set a series of masks and corresponding transfer mode types.	287
find	Finds a specific occurrence of a substring in an existing string or list element.	289
format	Creates a formatted string.	291
insertstring	Inserts a substring into an existing string.	293
loaddata	Loads the contents of a data element from a file, instead of setting them in a script via the data command.	295
logdata	Writes the content of a data element to the open log file.	296
lower	Forces all alpha characters in a given variable or list element to lower case.	297
mode	Sets the current transfer mode.	298
move	Initiates a move operation.	299
open	Opens a UDM session.	301
openlog	Opens a log file on disk for writing custom log information.	306
pad	Takes a string in an existing variable or list element and pads it to make it the given length.	308
parse	Parses a string, placing the components of the string into variables.	310
print	Prints a message in the UDM manager's transaction output.	312
query	Prints out the UDM Manager version.	313
quit	Exits the UDM Manager (same as the **`exit`** command).	314
rename	Renames a file.	315
replace	Replaces one or more instances of a sequence with another sequence.	316
report	Sets UDM's reporting options.	318
resetattribs	Resets the attributes for all UDM file systems on the transfer server with the specified logical name.	319
return	Stops executing the current script immediately and returns execution to the calling script immediately after the **`call`** command used to invoke the current script.	320
reverse	Reverses the order of all characters in the string of a specified existing variable or element.	321
savedata	Writes each line of a data element to a file on disk.	322

Name	Description	Page
set	Sets the UDM Manager's built-in and global variable values.	324
status	Displays the current connection status.	325
strip	Strips occurrences of a sequence from a string.	326
substring	Finds a substring in an existing string and stores it in a variable.	328
truncate	Truncates a string to a specific length.	330
upper	Forces all alpha characters in a given variable or list element to upper case.	331

Table 14.1 UDM Commands

14.2 appenddata

Syntax

```
appenddata data_element_name [value_1]... [value_n]
```

Description

The **appenddata** command appends a line of text to the end of an existing data element or, if that data element does not exist, creates a data element containing that line of text.

The *data_element_name* parameter specifies the name of a data element.

The *value_* parameters, which specify the text to be appended, are concatenated.

Variable references and expressions in these parameters are resolved, as with any other command, before assembling the line (with no spaces between each value) and appending it to the data element.

Parameters

Parameter	Description
data_element_name	Name of a data element.
value_1	First value in the line of text to be appended.
value_n	Last value in the line of text to be appended.

Table 14.2 **appenddata** Command Parameters

Examples

To append **The answer to 1 + 1 is 2** to the data element **mydata**:

```
appenddata mydata "The answer to 1 + 1 is" <1 + 1>
```

To append **SingleSystemImage** to the data element **mydata**:

```
appenddata mydata Single System Image
```

14.3 attrib

Syntax

```
attrib   logical-name[={dd|dsn|hfs|lib}]
         [attribute-name=[attribute_value]]...
```

Description

The **attrib** command sets the file system attributes that govern the transfer operations on the host with the specified logical name.

If only a logical name is specified in the **attrib** command, the current set of attributes for the specified host is displayed. For systems that support multiple file systems, such as z/OS and IBM i, the logical name can be followed by an file system name that indicates the files system to which the attribute applies:

• z/OS	hfs, dsn, dd
• IBM i	hfs, lib

If no file system name is specified, the attributes will be applied for the currently selected file system. If no attributes are specified, the transfer server returns its current set of attributes and their values.

Parameters

Parameter	Description
logical-name	Logical name of the transfer server for which to set the attributes (or from which to retrieve the attributes).
dd|**dsn**|**hfs**|**lib**	File system for which the attribute is to be set: • Values **dd** and **dsn** are valid only on z/OS file systems. • Value **hfs** is valid only on z/OS and IBM i file systems. • Value **lib** is valid only on IBM i file systems. If no file system is specified, the attribute is set for the current file system on the specified server.
attribute-name	Name of an attribute.
attribute-value	Value to be set for the attribute.

Table 14.3 **attrib** Command Parameters

Common File System Attributes

The following attributes are common to UDM on most platforms:

Attribute Name	Values	Description
createop	**append**, **new**, or **replace**	Specification for how the file is to be created: • If value is **append**, the transferred data is appended to any data already in an existing destination file. If the destination file does not exist, it is created. • If value is **new**, the UDM copies the file only if the destination file does not already exist. If the destination file does exist at the time the copy operation is initiated, the operation returns with an error. • If value is **replace**, UDM overwrites an existing destination file. Otherwise, UDM creates the file. [Default is **new**.]
defext	Any sequence of characters valid for the destination file system.	Sequence of characters appended to the end of the filename used to write the destination file, if the source filename is being used implicitly as the destination filename. This occurs after the file extension has been truncated (if **truncext** is set to **yes**). [By default, no default extension is defined.] Note: The sequence of characters is appended verbatim. UDM does not add a dot character before the sequence, so if one is desired, it must be specified explicitly.
eol	Any sequence of valid text data	End-of-line sequence used in text transfers. For the source side of a copy operation, excepting those from the z/OS **dd** and **dsn** file systems and the IBM i **lib** file system, the end-of-line sequence is used to determine the end of each line of data. When the specified sequence occurs in the data, UDM considers all data read up to that point (starting from the previous line) as a single line. Each line is transferred without the end-of-line sequence. On the destination side of a text transfer, the end-of-line sequence is appended to the end of each line before it is written. Two special character sequences can be used in any end of line sequence: • • **\r** sequence indicates a carriage return. • • **\n** sequence indicates a line feed. [Default depends on the platform and the file system: • Under Windows, the default is **\r\n**. • For UNIX platforms and the HFS file system under USS, the default value is \n. • Under z/OS (for the **dd** and **dsn** file systems) and IBM i (for the **lib** file system), the **eol** attribute is undefined. • Under IBM i (for the **hfs** file system), the default is **FILE**, which makes end-of-line terminator consistent with file **`ccsid`**.]

Attribute Name	Values	Description
linelen	A positive integer	Maximum length of each line of data (record under the z/OS **dd** and **dsn** file systems) written. It applies only to the destination side of a transfer and is used in conjunction with the **lineop** and **padline** attributes. [Default is 0.] • Under Windows, UNIX, and the **hfs** file system, this means no line operation takes place. • Under z/OS (for the **dd** and **dsn** file systems), the value **linelen** will be set equal to the logical record size used in allocating the destination file.
lineop	**none**, **stream**, **wrap**, or **trunc**	Line operation for transferred lines (records under the z/OS **dd** and **dsn** file systems). For the line operation to be in play in the transfer, the value of **linelen** must not be zero (**linelen** is set automatically for the z/OS **dd** and **dsn** file systems to the logical record size if it is zero). • If value is **none**, each source line or record or data is written as it is received as a complete line or record. If the length of the source line is greater than that specified by **linelen**, UDM issues an error and the transfer operation is aborted. • If value is **stream**, the source data is treated as one long, single line of data. The source data is broken into multiple lines (records), each with a length of that specified by **linelen**. • If value is **trunc**, each source line longer than the value specified by **linelen** is truncated to be exactly **linelen** characters long. • If value is **wrap**, each source line longer than the value specified by **linelen** is broken up into multiple lines, each no longer than **linelen** characters long. Each segment is written out as a separate line (record). Note: If an end of line sequence is specified, the length of the sequence is not considered by UDM when determining the length of a line on the destination side. UDM only looks at the raw data that is transferred. [By default, the **lineop** attribute is not defined.]
mode	Set of three numbers (0-7) or nothing	Note: This attribute is used only for UDM for UNIX. Specification for the mode (in UNIX parlance), or file permissions, of a file created by UDM in a copy operation. Existing files do not have their modes modified by UDM. They retain the file mode that they had before the copy operation was initiated. Each number in the set corresponds to one or more individuals for whom access is granted for the file: • First number Owner of the file. • Second number Users in the group assigned to the file. • Third number Everyone else. The value of each number is the sum of values representing file permissions: • 0 No permissions. • 1 Permission to execute the file. • 2 Permission to write to the file. • 4 Permission to read from the file. [By default, the mode attribute is not set. The default mode of a newly created file by UDM is dependent upon the user's umask or the mode of the source file in a UDM transfer.]

Attribute Name	Values	Description
padline	**none**, **null**, or **space**	Specification for whether or not data is padded. (Used in conjunction with **linelen**, when **linelen** is not zero.) • If value is **none**, each line (record) of data written is not padded. • If value is **null**, each line of data is padded with null characters (hex value 0) at the end until the line is **linelen** characters in length. • If value is **space**, each line of data is padded with spaces at the end until the line is **linelen** characters in length. [Default is **none**.]
srccreatetime	**yes** or **no**	Note: This attribute is used only for UDM for Windows. Specification for whether or not the creation timestamp of the destination file in a copy operation matches the creation timestamp of the source file. • If value is **yes**, the creation timestamp of the destination file matches the creation timestamp of the source file. • If the value is **no**, the creation timestamp of the destination file does not match the creation timestamp of the source file. [Default is **no**.]
srcmodtime	**yes** or **no**	Note: This attribute is used only for UDM for UNIX and Windows. Specification for whether or not the modification timestamp of the destination file in a copy operation matches the modification timestamp of the source file. • If value is **yes**, the modification timestamp of the destination file matches the modification timestamp of the source file. • If the value is **no**, the modification timestamp of the destination file does not match the modification timestamp of the source file. [Default is **no**.]
srcaccesstime	**yes** or **no**	Note: This attribute is used only for UDM for UNIX and Windows. Specification for whether or not the last access timestamp of the destination file in a copy operation matches the last access timestamp of the source file prior to the copy operation. • If value is **yes**, the last access timestamp of the destination file matches the last access timestamp of the source file prior to the copy operation. • If the value is **no**, the last access timestamp of the destination file does not match the last access timestamp of the source file prior to the copy operation. [Default is **no**.]
trans	**yes** or **no**	Specification for whether or not a transactional file copy is to be performed: • If value is **yes**, the file is copied to a file with a temporary name that is renamed to the destination filename once the file has been successfully transferred. • If value is **no**, the file is copied directly to the specified destination filename. [Default is **no**.] Note: For z/OS and IBM i, `trans` is valid only under the **hfs** file system.

Attribute Name	Values	Description
truncext	**yes** or **no**	Specification for whether or not the source filename's extension should be truncated if it is being used as the destination filename (no filename was explicitly specified on the destination side of the transfer operation). • If the value of this attribute is **yes**, the extension is truncated. • If the value is **no**, the filename is left untouched. [Default is **no**.] Note: UDM considers a file extension to be the sequence of characters following the last dot (**.**) character in the filename. When an extension is truncated, the dot marks the beginning of the extension and is truncated as well. UDM will not consider the a dot character as the first character in a filename as indicating a file extension.
umask	Three-digit octal value	File permissions mask used to create the destination file. Refer to the UNIX man page umask(1) for complete details. [By default, **umask** is not defined.] Note: For UNIX and z/OS (under the **hfs** file system), **umask** is valid only on the destination side of the transfer.
usefqn	**yes** or **no**	Specification, when copying a data set under the **dsn** file system, whether or not the fully qualified data set name is sent over as the source file name to be used by the destination if an explicit destination filename is not given. • If set to **yes**, the fully qualified data set name is sent. • If set to **no**, only the part of the data set name matching the source mask in the copy operation is used as the destination filename. [Default is **no**.] Note: For z/OS, **usefqn** is valid only on the source side of the transfer.

Table 14.4 Common File System Attributes

Example

To set the line length, line operation, and line padding sequence:

```
attrib ntmachine linelen=80 lineop=wrap padline=none
```

14.4 break

Syntax

```
break
```

Description

The **break** command stops iterating through a **forfiles** loop and picks up execution at the script line immediately following the **end** statement marking the end of the **forfiles** loop.

Example

To iterate through the files on a machine and attempt to delete each file (if the delete operation fails, UDM breaks out of the loop):

```
forfiles local=*
   delete $(_path)
   if $(_lastrc) NE 0
      print msg="Failed to delete $(_path)"
      break
   end
end
```

14.5 call

Syntax

```
call script-file [parameter-name=parameter-value]...
```

Description

The `call` command loads and executes a command script.

Scripts are interpreted line by line, with parameter substitution made just before each line is executed. Parameter substitution is indicated with a $(PARAMETER_NAME) sequence in the called script. Parameters are replaced by the value corresponding with the parameter's name (name=value format) in the `call` command itself. Each parameter of the `call` command must have a corresponding value.

`call` commands can be nested in called scripts up to ten levels deep.

If a parameter with the same name appears more than once in `call`, the first instance of the parameter with that name is used. If a parameter is referenced in the script, but was not passed in via the `call` command, an error is issued when the line with the reference is parsed.

Parameters

Parameter	Description
script-file	Filename of the script to process.
parameter-name	Name of a parameter to pass to the script.
parameter-value	Value to be set for the parameter.

Table 14.5 **call** Command Parameters

Examples

To invoke a script called **script.udm**.

- Parameter **file** has a value of *****.
- Parameter **src** has a value of **c:\source**.
- Parameter **dst** has the value **/etc/dest**.

```
call script.udm file=* src=c:\source dst=/etc/dest
```

To invoke a script under IBM i, the member name is required and can be ***FILE**:

```
call mylib/myfile(myscript)
```

Specifying ***FILE** invokes the normal default IBM i file search order.

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

The following example illustrates both:

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

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

14.6 cd

Syntax

```
cd logical-name[=directory]
```

Description

The **cd** command changes the working directory (on UNIX, Windows, IBM i, and file system HFS; on z/OS, the current data set qualifiers for DSN and DD file systems) on the specified logical machine (*logical-name*) to the specified path.

If no directory is specified, the current working directory (or qualifier) is printed to the UDM Manager.

Parameters

Parameter	Description
logical-name	Logical name of the transfer server to execute the **cd** command.
directory	Directory to be changed to the working directory on the server.

Table 14.6 **cd** Command Parameters

Example

To change the current directory on the machine with the logical name **ntmachine** to **c:\src**:

```
cd ntmachine=c:\src
```

14.7 close

Syntax

```
close
```

Description

The `close` command closes the current transfer session.

When a close message is issued, the UDM Manager lets the primary server know to close down the session. The primary server in turn notifies the secondary server.

Once a session has been closed, a new transfer session can be established with the **`open`** command. If a session is not open, an error message is printed.

14.8 closelog

Syntax

```
closelog
```

Description

The **`close`** command closes the open log file.

If the **`close`** command is issued when a log file is not open, an error is produced.

14.9 compare

Syntax

```
compare  STRING_1 STRING_2 [case=yes|no] [length=length]
```

Description

The **`compare`** command compares two strings.

The first two parameters (*STRING_1* and *STRING_2*) are the strings to be compared.

The optional **`case`** parameter specifies whether the comparison is case-sensitive (**yes**) or case-insensitive (**no**). [Default is **no**.]

If the **`length`** parameter is set, only the first **n** characters are compared.

The `_`**`lastrc.result`** variable receives the result of the comparison:

- If the strings match, 0 is stored in the result.
- If the strings do not match, the index of the point at which the comparison failed is stored as the result.

The `_`**`lastrc.message`** built-in variable contains MATCH if the strings are equal; it contains NO_MATCH if the strings are not equal.

Parameters

Parameter	Description
STRING_1	String to be compared to *STRING_2*.
STRING_2	String to be compared to *STRING_1*.
case=**yes** | **no**	Specification for whether or not the comparison is case-sensitive: • If set to **yes**, comparison is case-sensitive. • If set to **no**, comparison is not case-sensitive. Note: If **case** is not used, the comparison is case insensitive.
length=*length*	First **n** characters to be compared.

Table 14.7 **compare** Command Parameters

14.10 copy

Syntax

```
copy source-logical-name=source-file-specification
     [destination-logical-name=destination-file-specification]
```

Description

The **copy** command initiates a copy operation.

source-logical-name specifies the logical name of the source server (logical name of either the primary or secondary transfer server specified in the open command).

source-file-specification specifies the complete path or single file name of the file or files to be copied.

Optionally, *destination-logical-name* and *destination-file-specification* can be used to specify the logical name of the destination server and the complete path or single file name for the destination file.

If *destination-file-specification* identifies only a file name, the current directory (or high-level qualifier) is used for the destination server. If *destination-file-specification* identifies only a directory, the file name specified in *source-file-specification* is used.

If *destination-logical-name* and *destination-file-specification* are not specified, the other server in the transfer session (that is, the server not specified in *source-logical- name)* is assumed to be the destination server and *source-file-specification* is used for the destination file name.

By default, the destination file has timestamps matching its creation date, last modification date, and access date. However, via attributes set in the attrib command, you can set the destination file to have timestamps matching the source file.

Note: If an error is encountered, a copy operation will halt, and control will be returned to the script.

Parameters

Parameter	Description
source-logical-name	Logical name of the server acting as the source of the transfer operation.
source-file-specification	Complete path or single file name of the file or files to be copied. The file name (or file name portion of the path) can contain wildcard characters: • Wildcard ***** represents zero or more characters. • Wildcard **?** indicates a single character.
destination-logical-name	Logical name of the destination server in the transfer operation.
destination-file-specification	Complete path or single file name for the destination file.

Table 14.8 **copy** Command Parameters

Examples

To copy file **`test.txt`** - from a machine with logical name **`src`** to a machine with logical name **`dst`** - as **`test.bak`**:

```
copy src=test.txt dst=test.bak
```

To copy all files in the current directory from a machine with logical name **`src`** to the other machine in the transfer session:

```
copy src=*
```

14.11 copydir

Syntax

```
copydir  source-logical-name=file-specification
         [destination-local-name=file-specification]
```

Description

The **copydir** command initiates a copy operation that recurses into subdirectories.

If the source of the copy operation has subdirectories beneath the location given by the file-spec, UDM will create those directories on the destination side of the transfer and copy their contents as well. If any of the directories already exist on the destination side (relative to the destination file specification), the copy operation will fail if the **creatop** attribute on the destination side is not set to replace.

The source file specification is given as the value for source host's logical name (which should be either the primary or secondary logical name specified in the **open** command. An optional destination file-spec may be given as well. If none is given, the current directory (or high-level qualifier) for the destination machine in the transfer session is used.

If no destination server is given in the command (the command contains only the source's logical name and file specification), the other server in the transfer operation is assumed to be the destination and the source filename is used for the destination filename.

If a destination file specification is not given, or contains only a directory sequence, but no filename, the filename of the source file will be used.

The **copydir** command is available only on UNIX, Windows, and the **hfs** file systems for z/OS and IBM i.

Parameters

Parameter	Description
source-logical-name	Server acting as the source of the transfer operation.
file-specification	File or files to be copied. It also can specify a directory whose contents should be copied. The file specification can be a single filename, directory name, or a complete path to a file or directory. The filename (or filename portion of the path) can contain wildcard characters: • Wildcard ***** represents stands for zero or more characters. • Wildcard **?** indicates a single character.
destination-logical-name	Logical name of the destination server in the transfer.
file-specification	Complete path or filename for the destination file.

Table 14.9 **copydir** Command Parameters

Examples

To copy all files in a directory, recursing through all subdirectories:

```
copydir local=/mydir/*
```

To copy all files in a directory (same as above) by specifying the directory name only in the source (no wildcards or filename portion is needed):

```
copydir local=/mydir
```

To copy the files in **`mydir`** and all of its subdirectories into another existing directory on the destination side:

```
copydir local=/mydir dest=/yourdir
```

To copy an entire directory structure underneath a subdirectories and any files ending in **`.txt`**:

```
copydir local=/mydir/*.txt
```

14.12 data

Syntax

```
data [name|print=name] [resolve={all|defined|no}] data-element
     [end=end-sequence]
```

Description

The **`data`** command defines an in-stream data element that can be passed as input for other commands.

Parameters

Parameter	Description
name | **print**=*name*	Specifies either the **name** of the in-stream data element being defined (*name*) or a request to **print** the lines of that data element.
resolve={**all** | **defined** | **no**}	Variable resolution method: • **all** Resolve all variable references in the data. • **defined** Resolve only defined variable references in the data. • **no** Do not resolve any variable references in the data.
data-element	Contents of the in-stream data element being defined.
end=*end-sequence*	Sequence indicating the end of the data.

Table 14.10 **data** Command Parameters

14.13 debug

Syntax

```
debug    [EXPRESSION_SHOW_POSTFIX=yes|no]
         [EXPRESSION_SHOW_EVALUATION=yes|no]
         [COMMAND_SHOW_STRUCTURE=yes|no]
```

Description

The **debug** command turns debug information on and off.

Each parameter identifies a debug feature, as described in Table 14.11, and specifies whether that feature is turned on (**yes**) or off (**no**).

Parameters

Parameter	Description
EXPRESSION_SHOW_POSTFIX=**yes** | **no**	Postfix version of an expression after it has been converted from infix notation.
EXPRESSION_SHOW_EVALUATION=**yes** | **no**	Evaluation of an expression.
COMMAND_SHOW_STRUCTURE=**yes** | **no**	Different elements of a command after it has been parsed.

Table 14.11 **debug** Command Parameters

14.14 delete

Syntax

```
delete   logical-name=file-specification
```

Description

The **`delete`** command deletes a file (or series of files if *file-specification* contains any wildcards) from the transfer server with the corresponding logical name.

Parameters

Parameter	Description
logical-name	Server from which you want to delete the file(s).
file-specification	File or files to be deleted; it can be a single filename or a complete path to a file or directory. The filename (or filename portion of the path) can contain wildcard characters: • Wildcard ***** represents zero or more characters. • Wildcard **?** indicates a single character. Note: In z/OS, wildcards can be used on sequential data sets.

Table 14.12 **delete** Command Parameters

Examples

To delete all members of a PDS:

```
delete local='my.pds.name(*)'
```

To delete all files in a single directory level:

```
delete local=/mydir/*
```

To delete a single file:

```
delete local=myfile.txt
```

14.15 deletestring

Syntax

```
deletestring   variable_name pos=position |{startseq=sequence
               [startseqnum=number]} length=length |{endseq=sequence
               [endseqnum=number]} [case=yes|no]
```

Description

The **`deletestring`** command removes a substring from an existing string.

The first parameter, **`VARIABLE_NAME`**, is the name of an existing variable.

The beginning of the sequence to be deleted is indicated by either its starting position (one-based index, using the **`pos`** parameter) or as immediately following a particular occurrence of a character sequence (**`startseq`** specifies the sequence and the optional **`startseqnum`** specifies the occurrence number).

The end of the substring is determined by either specifying the length of the substring using the **`length`** parameter or giving a sequence that indicates the end of the substring (**`endseq`** specifies the ending sequence and the optional **`endseqnum`** specifies the occurrence number).

The optional **`case`** parameter specifies whether the comparisons of the sequences are case-sensitive (**yes**) or case-insensitive (**no**). [Default is **no**.]

The **`_lastrc.message`** built-in variable will contain:

- NO_MATCH if start or end sequences were specified and could not be matched
- INVALID_VALUE if the starting position or length are out of range
- SUCCESS if a sub-string was successfully deleted

Parameters

Parameter	Description
variable_name	Name of an existing variable.
pos=*position*	Starting position of sequence to be deleted (one based index).
startseq=*sequence*	Starting position of sequence to be deleted (following a specific character sequence).
startseqnum=*number*	Occurrence number of starting position of sequence to be deleted (following a specific character sequence).
length=*length*	Length of the substring to be deleted.
endseq=*sequence*	Ending position of sequence to be deleted (preceding a specific character sequence).
endseqnum=*number*	Occurrence number of ending position of sequence to be deleted (preceding a specific character sequence).
case=**yes** | **no**	Indicates whether or not the comparisons of the sequence are case-insensitive. [Default is **no**.]

Table 14.13 **deletestring** Command Parameters

Examples

The following examples illustrate how `deletestring` is used starting with a sample string called `mystring` with an initial value of `This is not just some sample text`:

```
set mystring="This is not just some sample text"
deletestring mystring pos=8 length=4
echo "$(mystring)"
This is just some sample text

set mystring="This is not just some sample text"
deletestring mystring startseq=" " startseqnum=2 endseq=" "
echo "$(mystring)"
This is just some sample text

set mystring="This is not just some sample text"
deletestring mystring startseq=" " startseqnum=2 length=21
echo "$(mystring)"
This is text
```

14.16 echo

Syntax

```
echo [parm_1]...[parm_n]
```

Description

The **echo** command writes text to standard out.

Parameters

Parameter	Description
parm_1	First parameter to echo.
parm_n	Other parameter(s) to echo.

Table 14.14 **echo** Command Parameters

Examples

To write:

`This is my message`:

Execute:

```
echo "This is my message"
```

14.17 echolog

Syntax

```
echolog [value_1]... [value_n]
```

Description

The **`echolog`** command writes text to the open log file.

If a log file is not open, **`echolog`** issues an error.

Parameters

Parameter	Description
value_1	First parameter to echo.
value_n	Other parameter(s) to echo.

Table 14.15 **echolog** Command Parameters

Examples

To write:

```
Tobeornottobe!Thatisthequestion.
```

Execute:

```
echolog To be or not to be! That is the question.
```

To write:

```
To be or not to be! That is the question.
```

Execute:

```
echolog "To be or not to be! That is the question."
```

To write:

```
1 + 1 = 2.
```

Execute:

```
echo "1 + 1 = " <1 + 1>
```

14.18 exec

Syntax

```
exec {logical-name | host-name} {cmd=command | cmdref=command-ref |
     stc=started-task} [user=userid] [pwd=password] [port=port]
     [codepage=codepage] [file=filename | xfile=filename [key=key]]
     [option=option] [mergelog=yes|no] [trace=yes|no]
     [input=data-element] [svropt=server-options]
     [stdout=data-element] [stderr=data-element]
```

Description

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

The first parameter of the **exec** command is either:

- Logical name (**logical-name**) of a transfer server (valid only if a transfer session has been established)
- Host name (**host-name**) of the machine on which you want to execute the command.

Note: You must have the UCMD Server and Universal Broker installed on the machine on which the command is to be executed.

The second parameter is the command type, which is either:

- **cmd** (command)
- **cmdref** (command reference)
- **stc** (started task)

For any of these three types, the value (*command*) is the remote command to be executed. (See the Universal Command 4.2.0 Reference Guide for more information about command types.)

UDM must authenticate a user on the remote machine in order to execute a command.

- If a logical name is specified in the first parameter, the **user** and **pwd** values are inherited from the same options specified in the open command for that logical name. These inherited values can be overridden by specifying them explicitly in the **exec** command.
- If a host name is specified in the first parameter, the **user** and **pwd** values must be specified explicitly in the **exec** command.

The **port** and **codepage** values are inherited from the UDM Manager's configuration file unless overridden explicitly in the call to the **exec** command.

- **port** specifies which port the Universal Broker is listening on for the remote machine.
- **codepage** specifies to which codepage the output of the remote command is translated.

The **user**, **pwd**, **port**, and **codepage** parameters can be stored in an external file instead of being specified explicitly in the **exec** command.

- If a plain text file is used, use the **file** parameter to specify the name of this file.
- If the file was encrypted with Universal Encrypt, use the **xfile** parameter to specify the name of this file.

If an encryption key other than the Universal Encrypt default was used, specify that key with the **key** parameter.

These parameters, and the format of the file containing these parameters, work exactly like the corresponding option in the **open** command.

The **option** parameter is used to pass options to the UCMD Server (see the SCRIPT_OPTIONS option for UCMD Manager in the Universal Command 4.2.0 Reference Guide for more details).

Two streams of data come back from the remote command. By default, output from standard out and standard error of the remote command are written to standard out and standard error by the UDM Manager (SYSPRINT and SYSOUT, respectively, under z/OS). The **mergelog** option can be set to yes if you want both output streams written to the UDM transaction log (standard out under UNIX, Windows, and IBM i; SYSPRINT under z/OS).

By default, if the UDM Manager is invoked with tracing turned on, tracing will be turned on in the Universal Command Manager when UDM invokes it via the **exec** command. Likewise, if **trace** is turned off in the UDM Manager, the Universal Command Manager is invoked with tracing turned off. You can override this behavior for the UCMD Manager invocation by setting the **trace** option in the call to the **exec** command.

There are some commands that require input from standard input. To provide this input, you must create a data element with the data command containing the input. Specifying the name of the data element with the **input** parameter will cause the information in the data element to be sent over as standard input to the remote command.

The **svropt** parameter can be used to override UCMD Server options.

Note: UDM does not require a space before the server options, as does Universal Command.

The **stdout** and **stderr** parameters specify data elements to contain standard out and standard error, respectively, from the remote command. If the data elements do not exist, they are created. If the data elements do exist, they are overwritten with the output from the remote command. If the value portion refers to an existing non-data element variable or the name of a built-in variable (that is, any variable beginning with an underscore), an error is issued.

The **exec** command output will still be written to UDM stdout (the transaction log) and UDM stderr, where appropriate, even with the presence of the **stdout** and/or **stderr**.

Parameters

Parameter	Description
logical-name	Logical name of a transfer server as specified in the **open** statement (valid only if a transfer session has been established) of the machine on which you want to execute the command. Note: You must have the Universal Command Server and Universal Broker installed on the machine on which the command is to be executed.
host-name	Host name of the machine you wish to execute the command on. Note: You must have the Universal Command Server and Universal Broker installed on the machine on which the command is to be executed.
cmd=*command*	Command type **cmd** (command) will be executed on the remote server with its value being the command to be executed. (See the Universal Command Reference Guide for more information about available command types for each platform.)
cmdref=*command-ref*	Command type **cmdref** (command reference) will be executed on the remote server with its value being the command reference to be executed. (See the Universal Command Reference Guide for more information about available command types for each platform.)
stc=*started-task*	Command type **stc** (started task) will be executed on the remote server with its value being the started task to be executed. (See the Universal Command Reference Guide for more information about available command types for each platform.)
user=*user-id*	User ID (**user**) and password (**pwd**) are inherited from the same parameters specified in the **open** command for that logical name. These values can be overridden by specifying them explicitly in the **exec** command. Note: If a host name is used (instead of a logical name), a user ID and password must be specified explicitly in the **exec** command.
pwd=*password*	User ID (**user**) and password (**pwd**) options are inherited from the same options specified in the **open** command for that logical name. These values can be overridden by specifying them explicitly in the **exec** command. Note: If a host name is given (instead of a logical name), a user ID and password must be specified explicitly in the **exec** command.
port=*port*	Port that the Universal Broker is listening on for the remote machine. The **port** is inherited from the UDM Manager's configuration file unless explicitly overridden in the call to the **exec** command.
codepage=*codepage*	Codepage to which the remote command output is translated. The **codepage** is inherited from the UDM Manager's configuration file unless overridden explicitly in the call to the **exec** command.
file=*filename*	Plain text file containing the values for the remote execution server: **port**, **user**, **pwd**, and **codepage**. Note: These values override any corresponding values specified for the transfer server in the **open** command.
xfile=*filename*	Universal Encrypted text file containing the values for the remote execution server: **port**, **user**, **pwd**, and **codepage**. Note: These values override any corresponding values specified for the transfer server in the **open** command.

Parameter	Description
key=*key*	Key used to decrypt the file specified by the **xfile** parameter. If this parameter is not present, the default key for Universal Encrypt is used.
option=*option*	Passes options to the Universal Command Server.
mergelog=**yes** | **no**	Specification for whether or not to merge the two streams of data that come back from the remote command to the UDM transaction log (SYSOUT under UNIX, Windows, and IBM i; SYSPRINT under z/OS). By default, output from standard out and standard error of the remote command are written to standard out and standard error by the UDM Manager (SYSPRINT and SYSOUT, respectively, under z/OS). [Default is **no**.]
trace=**yes** | **no**	Overrides the trace behavior of the Universal Command Manager when invoked via the **exec** command. [Default is the trace value of the UDM Manager.]
input=*data-element*	Data element to be used as standard input to the remote command.
svropt=*server-options*	Overrides Universal Command Server options.
stdout=*data-element*	Data element to be used as standard out from the remote command.
stderr=*data-element*	Data element to be used as standard error from the remote command.

Table 14.16 **exec** Command Parameters

14.19 execsap

Syntax

```
execsap  host={host-name|destination} type={event|generic}
         [eventid=event-id] [parm=event-parm] [client=client]
         [user=userid] [pwd=password] [codepage=codepage]
         [file=filename | xfile=filename [key=key]] [mergelog=yes|no]
         [trace=yes|no]
```

Description

The **execsap** command executes SAP events if you have a licensed version of the Universal Connector (version 3.1.1 or later) on the same system with the UDM Manager.

Note: UDM does not support the **execsap** command for IBM i and Windows.

The first parameter of the **execsap** command is either:

- Host parameter with an SAP destination entry
- Name of a destination in your SAP RFC file

The **type** parameter specifies the type of action being performed. A specified type of **event** requires that an SAP event ID be specified with the **eventid** parameter.

Note: For version 4.2.0, the only valid type is **event**, which triggers an SAP event.

An event parameter can be passed to the SAP event using the **parm** parameter. The **client** parameter specifies the SAP client.

UDM must authenticate a user SAP in order to execute an SAP action. The user id and password can be specified with the **user** and **pwd** parameters, respectively.

The **codepage** value is inherited from the UDM Manager's configuration file unless overridden explicitly in the call to the **execsap** command. **codepage** specifies to which codepage the output of the remote command is translated.

The **user**, **pwd**, and **codepage** parameters can be stored in an external file instead of being specified explicitly in the **execsap** command syntax.

- If a plain text file is used, the **file** parameter specifies the name of this file.
- If the file was encrypted with Universal Encrypt, the **xfile** parameter specifies the name of this file.

If an encryption key other than Universal Encrypt's default was used, that key can be specified with the **key** parameter.

These options and the format of the file containing these options work exactly like the corresponding option in the **open** command.

Two streams of data come back from the SAP execution. By default, output from standard out and standard error is written to standard out and standard error by the UDM Manager (SYSPRINT and SYSOUT, respectively, under z/OS). The **mergelog** parameter can be set to **yes** if you want both output streams written to the UDM transaction log (standard out under UNIX, Windows, and IBM i; SYSPRINT under z/OS).

By default, if the UDM Manager is invoked with tracing turned on, tracing will be turned on in the Universal SAP Connector when UDM invokes it via the **execsap** command. Likewise, if **trace** is turned off in the UDM Manager, the Universal SAP Connector is invoked with tracing turned off. You can override this behavior for the Universal SAP Connector invocation by setting the **trace** option in the call to the **execsap** command.

Parameters

Parameter	Description
host=*hostname* | *destination*	SAP destination entry or, if host=name format is used, the name of a destination in you SAP RFC file.
type=**event** | **generic**	Type of action (**event** is the only type supported under UDM 4.2.0).
eventid=*event-id*	ID of the SAP event to trigger.
parm=*event-parm*	Passes an event parameter to the SAP event.
client=*client*	SAP client for which to perform the action.
user=*user-id*	User ID (**user**) and password (**pwd**) are inherited from the same parameters specified in the **open** command for that logical name. These values can be overridden by specifying them explicitly in the **execsap** command. Note: If a host name is used (instead of a logical name), a user ID and password must be specified explicitly in the **execsap** command.
pwd=*password*	User ID (**user**) and password (**pwd**) are inherited from the same parameters specified in the **open** command for that logical name. These values can be overridden by specifying them explicitly in the **execsap** command. Note: If a host name is used (instead of a logical name), a user ID and password must be specified explicitly in the **execsap** command.
codepage=*codepage*	Codepage to which the remote command output is translated. The **codepage** is inherited from the UDM Manager's configuration file unless overridden explicitly in the call to the **execsap** command.

Parameter	Description
file=*filename*	Plain text file containing the values for the remote execution server: **user**, **pwd**, and **codepage** These values override any corresponding values specified for the transfer server in the **open** command.
xfile=*filename*	Universal Encrypted text file containing the values for the remote execution server: **port**, **user**, **pwd**, and **codepage**. These values override any corresponding values specified for the transfer server in the **open** command.
key=*key*	Key used to decrypt the file specified by the **xfile** parameter. If this parameter is not present, the default key for Universal Encrypt is used.
mergelog=**yes** | **no**	Specification for whether or not to merge the two streams of data that come back from the remote command to the UDM transaction log (SYSOUT under UNIX, Windows, and IBM i; SYSPRINT under z/OS). By default, output from standard out and standard error of the remote command are written to standard out and standard error by the UDM Manager (SYSPRINT and SYSOUT, respectively, under z/OS). [Default is **no**.]
trace=**yes** | **no**	Overrides the trace behavior of the Universal Command Manager when invoked via the **execsap** command. [Default is trace value of the UDM Manager.]

Table 14.17 **execsap** Command Parameters

14.20 exit

Syntax

```
exit
```

Description

The **`exit`** command behaves the same as the **`quit`** command (see Section 14.37 quit).

14.21 filesys

Syntax

filesys *logical-name*[={**dd**|**dsn**|**hfs**|**lib**}]

Description

The **filesys** command sets the file system with which the server specified by **logical-name** is working.

UNIX and Windows

For UNIX and Windows systems, there is only one file system; specifying a **filesys** value will result in a warning.

z/OS

For z/OS systems, this value can be either:

- dd (ddnames defined with JCL DD statements)
- dsn (data set name)
- hfs (UNIX System Services file system).

Note: dd is available only on z/OS Manager for two-party transfer.

IBM i

For IBM i systems, this value can be either:

- lib (Library file system)
- hfs (IFS file system: root or QOpenSys)

Parameters

Parameter	Description
logical-name	Logical name of the transfer server on which to change the file system.
dd | **dsn** | **hfs** | **lib**	File system with which the specified server is working: • If the value is **dd**, the DD name file system is used on the specified server (valid only for z/OS). • If the value is **dsn**, the data set name file system is used (valid only for z/OS). • If the value is **hfs**, the HFS file system is used. (Valid only for z/OS and IBM i. For z/OS, USS is used; for IBM i, IFS (root or QOpenSys) is used.) • If the value is **lib**, the **LIB** file system is used (valid only for IBM i). If no value is given, UDM will return the current file system for the specified transfer server. [Default is **dsn**.] Note: UDM supports multiple file systems only under z/OS and IBM i. An attempt to change the file system for a server running under Windows or UNIX will result in an error.

Table 14.18 **filesys** Command Parameters

Example

To set the file system on the server with the logical name **`mvsmachine`** to the **`dsn`** file system:

```
filesys mvsmachine=dsn
```

14.22 filetype

Syntax

```
filetype    [binary|text=file-mask-1]...[binary|text=file-mask-n]
            [remove=file-mask-1]...[remove=file-mask-n] [resetall]
```

Description

The **filetype** command sets a series of masks and corresponding transfer mode types.

For any file whose source name matches a specific mask, the transfer mode type corresponding to that mask (text or binary) is used in transferring that file. If the source file name in a transfer does not match any registered masks, the default transfer mode type (set using the **mode** command) is used.

Issuing the **filetype** command by itself, with no parameters, dumps all of the file masks and their corresponding mode types that have been set.

Parameters

Parameter	Description
binary | **text** = *filemask_1*	Sets the specified file mask (*filemask_1*) so that source file names matching this mask will be transferred as binary files.
binary | **text** = *filemask_n*	Sets the specified file mask (*filemask_n*) so that source file names matching this mask will be transferred as binary files.
remove=*filemask_1*	Removes an entry that matches the specified file mask (*filemask_1*).
remove=*filemask_n*	Removes an entry that matches the specified file mask (*filemask_n*).
resetall	Removes all file mask entries.

Table 14.19 **filetype** Command Parameters

Examples

To set a single file mask and its corresponding transfer mode type:

```
filetype binary=*.exe
```

To set a series of file masks in a single call, instead of making multiple calls:

```
filetype text=*.txt text=*.c binary=*.exe binary=*.dat
```

To remove a file mask:

```
filetype remove=*.txt
```

14.23 find

Syntax

```
find string seq=sequence [pos=index] [case=yes|no] [num=number|last]
```

Description

The **find** command finds a specific occurrence of a substring in an existing string or list element.

The first parameter, *string*, is the string in which to search for the sequence. It can be a variable reference or a literal. The **seq** parameter specifies the sequence for which the search is being made.

The optional **pos** parameter specifies the one-based index of the string where the find operation begins.

The optional **case** parameter indicates whether the search is case-sensitive (**yes**) or case-insensitive (**no**). [Default is **no**.]

The optional **num** parameter specifies the instance of the sequence for which the search is being made. If the value of **num** is **last**, the find function gives back the index of the last occurrence of the sequence in the string.

If the sequence is found, the index (one-based) of the starting point of the requested occurrence is place in the **_lastrc.result** variable. If the sequence was not found in the string, a value of -1 is placed in the **_lastrc.result** variable.

If the sequence is found, **_lastrc.message** contains a value of MATCH. If the sequence could not be found, **_lastrc.message** contains NO_MATCH.

Parameters

Parameter	Description
string	String in which to search for the sequence.
seq=*sequence*	Sequence for which the search is being made.
pos=*index*	One-based index of the string where the `find` operation begins.
case=**yes** | **no**	Specification of whether the search is case-sensitive (**yes**) or case-insensitive (**no**). [Default is **no**.]
num=*number* | **last**	Instance of the sequence for which the search is being made.

Table 14.20 **find** Command Parameters

Examples

The following examples demonstrate the `find` command:

```
find "This is a test" seq=is
echo $(_lastrc.result)
3

find "This is a test" seq=" " num=2
echo $(_lastrc.result)
8

set mystring="I hate examples"
find "$(mystring)" seq=EXAMPLES case=yes
echo $(_lastrc.result)
-1

find "This is a test" seq=" " pos=6
echo $(_lastrc.result)
8
```

14.24 format

Syntax

```
format variable_name [{string_1|expression_1}
       [align={center|left|right|justify}] [pad=sequence]
       [trunc=yes|no] [length=length]]...[{string_n|expression_n}
       [align=center|left|right|justify] [pad=sequence]
       [trunc=yes|no] [length=length]]
```

Description

The **format** command creates a formatted string.

The first parameter, *variable_name*, is the variable or list element in which the newly formatted string is stored.

The second parameter, *string_1* (or *expression_1*), is the first section of the string. Each additional, optional parameter specifies formatting that affect that string.

All following parameters, *string_n* (or *expression_n*), are the next sections of the string. Each additional, optional parameter specifies formatting that affect that string.

Parameters

Parameter	Description
variable_name	Variable or list element in which the newly formatted string is stored
string_1 | *expression_1*	First section of the string
align=**center** | **left** | **right** | **justify**	Method of how the value is aligned within the given space. [Default alignment is left.]
pad=*sequence*	Character(s) used to pad out the formatted value. [Default is space character.]
trunc=**yes** | **no**	Truncates the value to the length given by `length` if it is longer than that value. [Default is to not truncate.]
length=*length*	Length of the formatted field. [Default is length of the value specified.]

Table 14.21 **format** Command Parameters

Examples

The following examples demonstrate the **format** command.

```
set firstname="Stonebranch"
set lastname="Incorporated"
format mystring "First Name: " "$(firstname)"  " Last Name: " +
"$(lastname)" length=3 trunc=yes
echo "$(mystring)"

Output:
First Name: Stonebranch Last Name: Inc


format mystring "The " "Value " "Is: " 4 length=8 align=right pad=0
echo "$(mystring)"
The Value Is: 00000004
```

14.25 insertstring

Syntax

```
insertstring  variable_name sequence {pos=position | startseq=sequence
startseqnum=number} [case=yes|no]
```

Description

The **insertstring** command inserts a substring into an existing string.

The first parameter, *variable_name*, is the name of the existing variable or list element into which to insert the sequence.

The second parameter, *sequence*, specifies the sequence to be inserted.

The insertion point of the sequence is either:

- By position (specified by the **pos** parameter)
- At the character immediately following the instance of a given sequence (**startseq** specifies the sequence and **startseqnum** gives the instance).

The optional **case** parameter specifies whether the comparison used to find the start sequence is case-sensitive (**yes**) or case-insensitive (**no**). [Default is **no**.]

If the start sequence is given and could not be found, **_lastrc.message** will contain NO_MATCH. If a position is given and invalid, **_lastrc.message** will contain INVALID_VALUE.

Parameters

Parameter	Description
variable_name	Name of the existing variable or list element into which to insert the sequence.
sequence	Sequence to be inserted.
pos=*position*	Position at which sequence is inserted.
startseq=*sequence*	Sequence after which specified sequence is inserted.
startseqnum=*number*	Instance of sequence after which specified sequence is inserted.
case=**yes** | **no**	Specification of whether or not the comparison used to find the start sequence is case-insensitive. [Default is **no**.]

Table 14.22 **insertstring** Command Parameters

Example

The following examples show how the **`insertstring`** command can be used on a preexisting variable called **`mystring`** with a starting value of **`This is a string`**:

```
insertstring mystring " sample" pos=10
echo "$(mystring)"
This is a sample string

insertstring mystring "sample " startseq=" " startseqnum=3
echo "$(mystring)"
This is a sample string
```

14.26 loaddata

Syntax

```
loaddata data-element-name=file-path
```

Description

The **`loaddata`** command loads the contents of a data element from a file, instead of setting the contents in a script using the data command.

Parameters

Parameter	Description
data-element-name	Data element into which the file contents is loaded.
file-path	File from which contents is loaded into the data element.

Table 14.23 **loaddata** Command Parameters

Examples

If a file called **`commands.txt`** has the following contents:

```
cd /
ls -al
exit
```

Issuing the following commands would load the contents of **`commands.txt`** into a data element called **`mydata`** and print them out:

```
> loaddata mydata=commands.txt
> data print=mydata
cd /
ls -al
exit
```

14.27 logdata

Syntax

```
logdata data_element_name
```

Description

The **`logdata`** command writes the contents of a data element to the open log file.

If a log file is not open, or the named data element does not exist, an error is issued. This is effectively the same as constructing a **`fordata`** loop and writing out each line of the data element with the echolog command.

Any variable references appearing in the contents of the data element will follow the same resolution rules as if the operation were performed using the aforementioned **`fordata`**/echolog loop.

Parameters

Parameter	Description
data_element_name	Name of the data element to write to the open log.

Table 14.24 **logdata** Command Parameters

Example

To load some information into a data element, execute:

```
loaddata mydata=secretcodes.txt
```

To write that data element to the open log file, execute:

```
logdata mydata
```

14.28 lower

Syntax

```
lower variable_name
```

Description

The `lower` command forces all alphabetic characters in a given variable or list element to lower case.

Parameters

Parameter	Description
variable_name	Variable or list element in which to force alphabetic characters to lower case.

Table 14.25 **lower** Command Parameters

14.29 mode

Syntax

```
mode [type={text|binary}] [trim={yes|no}]
```

Description

The **`mode`** command sets the current transfer mode.

The **`type`** parameter specifies the current mode type:

- If **`text`** is the current mode type, file transfers are treated as text and codepage translation takes place.
- If **`binary`** is the current mode type, files are transferred as binary.

The **`trim`** parameter specifies whether or not trailing spaces are trimmed (removed) from each record / line in a text transfer.

Issuing the **`mode`** command without parameters displays the current transfer mode.

Parameters

Parameter	Description
type=**text** | **binary**	Sets the type of transfer. • If the transfer type is **text**, UDM performs text translation on the data between the codepages for the primary and secondary servers. • If the transfer type is **binary**, UDM sends the source data over as is, with no translation. [Default is **binary** when a new session is opened.]
trim=**yes** | **no**	Sets whether or not to trim trailing spaces during a text transfer. • If the value is **yes**, spaces at the end of each line (record) are removed from the data when it is written. • If the value is **no**, spaces at the end of each line (record) are not removed. [Default is **no** when a transfer session is initiated.]

Table 14.26 **mode** Command Parameters

Example

To set the current transfer type to text transfers:

```
mode type=text
```

14.30 move

Syntax

```
move  source-logical-name=source-file-specification
      [destination-logical-name=destination-file-specification]
```

Description

The **move** command initiates a move operation.

A move operation is similar to a copy operation. The only difference between a **move** command and a copy command is that after you move a file, it is deleted from the source server from which it was moved.

If the **move** command cannot delete the source file after the move, the move operation has failed; **move** deletes the destination file.

z/OS

If the source side of the move is in the DD file system, or if the source is a GDG, an error will be issued specifying that the **move** command is not supported.

source-logical-name specifies the logical name of the source server (logical name of either the primary or secondary transfer server specified in the open command).

source-file-specification specifies the complete path or single file name of the file or files to be copied.

Optionally, *destination-logical-name* and *destination-file-specification* can be used to specify the logical name of the destination server and the complete path or single file name for the destination file.

If *destination-file-specification* identifies only a file name, the current directory (or high-level qualifier) is used for the destination server. If *destination-file-specification* identifies only a directory, the file name specified in *source-file-specification* is used.

If *destination-logical-name* and *destination-file-specification* are not specified, the other server in the transfer session (that is, the server not specified in *source-logical- name)* is assumed to be the destination server and *source-file-specification* is used for the destination file name.

Note: As with a copy command, if an error is encountered, a move operation will halt, and control will be returned to the script.

Parameters

Parameter	Description
source-logical-name	Logical name of the server acting as the source of the move operation.
file-specification	File or files to be copied. It can be a single filename or a complete path. The filename (or filename portion of the path) can contain wildcard characters: • Wildcard ***** represents for zero or more characters. • Wildcard **?** indicates a single character.
destination-logical-name	Logical name of the destination server in the move operation.
file-specification	Complete path or file name for the destination file.

Table 14.27 **move** Command Parameters

Examples

To move file **`test.txt`** - from a machine with logical name **`src`** to a machine with logical name **`dst`** - as **`test.bak`**:

```
move src=test.txt dst=test.bak
```

To move all files in the current directory from a machine with logical name **`src`** to the other machine in the transfer session:

```
move src=*
```

14.31 open

Syntax

```
open [primary={*|local|host-name}
        [port=broker-port] [user=username [pwd=password]]
        [codepage=codepage] [{file=filename | xfile=filename
        [key=key]}]
     secondary=host-name [port=broker-port] [user=username
        [pwd=password]] [codepage=codepage] [{file=filename |
        [xfile=filename [key=key]}]
     [encrypt={yes|no|cipher}]
     [compress=compression-method]
     [nft=yes|no]
     [comment=text]
```

Description

The **open** command opens a UDM transfer session.

Each transfer session has a primary transfer server and a secondary transfer server. These servers are given logical names (*primary* and *secondary*) by the user.

Each logical name is set to the host name or IP address (*host-name*) of the transfer server. For a two-party transfer session, the UDM Manager acts as the primary server, and its logical name (*primary*) must be set to `*` or `local` as the host name.

Optionally, only a *secondary* server may be specified, which implies a two-party transfer. In this case, the primary server automatically is assigned `local` as the logical name.

Each transfer server parameter (*primary* and *secondary*) can be followed by one or more of the following parameters that further define the transfer set-up (each parameter applies to the transfer server which it immediately follows):

- **port** specifies the port that the broker is accepting requests on to start a UDM server.
- **user** specifies the user (local to the host on which the server will be running) under which the transfer operation is being carried out
- **pwd** is the password for the user.
- **codepage** specifies the codepage that will be used for text translation of transferred data.

Note: The **user** and **pwd** parameters are not required for the local side (primary server) of a two-party transfer, as the UDM Manager will be running as the user that invoked it.

Each transfer server parameter also can be followed by the **`file`** or **`xfile`** parameter. These parameters specify a file (plain text or Universal Encrypted text, respectively) that contain **`port`**, **`user`**, **`pwd`**, and/or **`codepage`** in the format of a UDM command file (see file / xfile Parameters Format in Examples). If a file is specified, its values apply to the transfer server that the **`file`** or **`xfile`** parameter follows. These file values override any values specified by the **`port`**, **`user`**, **`pwd`**, and /or **`codepage`** parameters following that transfer server.

The optional **`key`** parameter specifies the encryption key used to decrypt the encrypted file specified by **`xfile`**.

The **`encrypt`** parameter specifies either:

- **yes**
  An agreed-upon cipher will be negotiated based on the components **`data_ssl_cipher_list`** configuration value.
- **no**
  NULL-MD5 is used as the encryption method.
- *cipher*
  Specific cipher to use as encryption method: RC4-SHA, RC4-MD5, AES256-SHA, AES128-SHA, DES-CBC3-SHA, DES-CBC-SHA, NULL-SHA, or NULL-MD5. Specifying NULL-NULL as the cipher completely disables SSL when NULL-NULL also is specified in the UDM Server Data Cipher Lists associated with a transfer.

Note: If **`encrypt`** is not used, the value specified by the UMD Manager ENCRYPT option is used (default is **no**).

The **`compress`** parameter can have either of the following values:

- **yes**
  Compression option in the UDM Manager's configuration file is used.
- **no**
  No compression is used when transferring data.
- **zlib**
  Forces the transfer servers to use ZLIB compression when transferring files.
- **hasp**
  Forces the transfer servers to use HASP compression.

If **`compress`** is not specified, a default value of **no** is used.

The **`nft`** parameter specifies whether or not the UDM sessions will be network fault tolerant.

The **`comment`** parameter specifies a comment for a single session (or overrides a comment specified by the COMMENT option.)

For example: **`open src=* dst=zos14 comment="Data transfer for account 94882"`**

Parameters

Parameter	Values	Description
primary	**[{*|local|***host name***}]**	Logical name of the primary transfer server. If the value is * or **local**, a two-party transfer is initiated with the UDM Manager acting as the primary server. Otherwise, a three-party transfer session is initiated with the primary transfer server running on the machine specified by *host name*. If no primary transfer server is specified, a two-party transfer session is implied, with **local** as the UDM Manager / primary server's logical name.
secondary	*host name*	Logical name of the secondary transfer server. Its value is the host name or IP address of the machine on which the server will be running. Note: The host name of the secondary transfer server must be given from the perspective of the primary transfer server, not the UDM Manager.
port *	TCP port number or service name	Port on which the broker that will initiate the transfer server is listening. If this parameter is not used, the port number is assumed to be **7887**. Note: The option is not valid for the primary server in a two-party transfer.
user *	Valid username on the system the transfer server will be running on	User name to authenticate with on the transfer server. The user name must be valid on the system. Once authenticated, the default directory on the transfer server is set to the user's home directory under UNIX and HFS. Under Windows, the default directory will be a directory created for the user underneath where the Stonebranch Solutions suite is installed. For z/OS under the **dsn** file system, the user name will be the high level qualifier. Note: This option is not valid for the primary server in a two-party transfer.
pwd *	Password of the user to authenticate.	Password, for the specified user name, for authenticating the user on the transfer server. Note: This option is not valid for the primary server in a two party transfer.
codepage *	Valid codepage	Codepage used for text translation on the transfer server. If no codepage is specified, the codepage listed in UDM's configuration will be used.
file *	Valid filename	Plain text file containing the values for the transfer server: **port**, **user**, **pwd**, and/or **codepage** (see file / xfile Parameters Format in Examples). These values override any values specified by the **port**, **user**, **pwd**, and /or **codepage** parameters for the specified transfer server.
xfile *	Valid filename	Universal Encrypted text file containing the values for the transfer server: **port**, **user**, **pwd**, and/or **codepage** (see file / xfile Parameters Format in Examples). These values override any values specified by the **port**, **user**, **pwd**, and /or **codepage** parameters for the specified transfer server.
key *	Key used to decrypt the file specified by **xfile**	Key used to decrypt the file specified by the **xfile** parameter. If this parameter is not used, the default key for Universal Encrypt is used.

Parameter	Values	Description
encrypt	**yes**, **no**, or *cipher*	Encryption method for the transfer session. • If the value is **yes**, an agreed-upon cipher will be negotiated based on the components **`data_ssl_cipher_list`** configuration value. • If the value is **no**, the **NULL-MD5** cipher is used. Otherwise, a valid cipher must be specified: RC4-SHA, RC4-MD5, AES256-SHA, AES128-SHA, DES-CBC3-SHA, DES-CBC-SHA, NULL-SHA, or NULL-MD5. Specifying NULL-NULL as the cipher completely disables SSL when NULL-NULL is also specified in the UDM server Data Cipher Lists associated with a transfer.
compress	**yes**, **no**, **hasp**, or **zlib**	Compression method for the transfer session: • If the value is **yes**, the compression method specified in the UDM Manager's configuration is used. • If the value is **no**, no compression is used. • If the value is **hasp**, **HASP** compression is used. • If the value is **zlib**, **ZLIB** (ZIP) compression is used.
nft	**yes** or **no**	Specification for whether or not the session is network fault tolerant: • **yes** specifies that the session will be network fault tolerant. • **no** specifies the session will not be network fault tolerant.
* These parameters apply to the transfer server (primary or secondary) that they follow in the **open** command.		

Table 14.28 **open** Command Parameters

Examples

To open a two-party transfer session between two machines, explicitly specifying the primary server:

```
open primary=* secondary=mvsmachine
```

To open a two-party transfer session between two machines, not specifying the primary server:

```
open secondary=mvsmachine
```

To open a three-party transfer session between two machines:

```
open primary=ntmachine secondary=mvsmachine
```

To open a three-party transfer session between two machines with the primary server's options coming from an encrypted configuration file and the secondary server having an authenticate user and changing the codepage for its side of the transfer:

```
open primary=ntmachine xfile=enc.dat dst=test.bak user=me pwd=mypwd
     codepage=IBM277
```

file / xfile Parameters Format

The parameters in a **`file`** or **`xfile`** are in the same format as the parameters in a UDM command file, as shown in Table 14.29, below.

Note: **`file`** and **`xfile`** can be shared with Universal Command.

open Command Parameter Format	file / xfile Parameter Format	Description
port=*broker-port*	-port *broker-port*	Port that the broker is accepting requests on to start a UDM server
user=*user-name*	-userid *user-name*	User (local to the host on which the server will be running) under which the transfer operation is being carried out
pwd=*password*	-pwd *password*	Password for the user
codepage=codepage	-codepage *codepage*	Codepage that will be used for text translation of transferred data

Table 14.29 **`file`** / **`xfile`** Parameters Format

14.32 openlog

Syntax

`openlog` *`log_file_path`*`[append=`**`yes`**`|`**`no`**`]`

Description

The **`openlog`** command opens a log file on disk for writing custom log information.

Windows

The log file is open shared so that other processes can read it from while it is still open by UDM.

z/OS

The log file must be allocated as part of the job. The log file name takes the format of **`dd:ddname`**.

IBM i

The log file must be a LIB file system file; **`openlog`** does not support the HFS file system.

Only one log file can be open at any one time in UDM. If the user issues an **`openlog`** command while a log file is open, the user receives an error. The user can explicitly close the open log file by issuing a closelog command; otherwise the log file will be closed automatically when the manager ends.

UNIX, Windows

An optional **append** parameter indicates whether or not an existing log file should be appended to.

- If **append** is set to **YES** and the log file exists, any log statements written are appended to the end of that file.
- If **append** is set to **NO** and the log file exists, the file is truncated to zero length as part of the open operation.
- If **append** is set to either **YES** or **NO**, and the log file does not exist, it will be created.
- If **append** is not included in an issued openlog command, the log file is opened as if **append** were set to **NO**.

Parameters

Parameter	Description
log_file_path	Pathname of the log file to open.
append=**yes** | **no**	Specification for whether or not to append log statements to an existing log file.

Table 14.30 **openlog** Command Parameters

Examples

To open a log file under z/OS, execute:

```
openlog dd:mylog
```

To open a log file with an absolute path, execute:

```
openlog "c:\document and settings\user\logs\mylog.txt"
```

To open a log file with a relative path, execute:

```
openlog logs/mylog.txt
```

To open a log file so that new entries are appended to the end of the existing data in the file, execute:

```
openlog mylog.txt append=yes
```

14.33 pad

Syntax

```
pad variable_name length=length [seq=sequence]
```

Description

The **pad** command takes a string in an existing variable or list element and pads it to make it the given length. If the string already is longer than the specified length, it is not padded.

The first parameter, *variable_name*, is the name of an existing variable or list element to pad. The `length` parameter specifies the desired length of the padded string.

The optional `seq` parameter specifies the sequence that is used for padding the string; if `seq` is not specified, a space is used.

Parameters

Parameter	Description
variable_name	Name of an existing variable or list element to pad
length=*length*	Desired length of the padded string.
seq=*sequence*	Sequence used for padding the string.

Table 14.31 **pad** Command Parameters

Examples

The following examples use a predefined string, **`mystring`**, with a starting value of **`This is a test`**.

```
pad mystring length=20
echo "$(mystring) TheEnd"
This is a test      TheEnd
```

```
pad mystring length=20 seq="-"
echo "$(mystring)"
This is a test------
```

14.34 parse

Syntax

```
parse string_name var_1 [seq_1 var_2]... [seq_n var_n]
```

Description

The **parse** command parses a string, placing components of the string into variables.

The first parameter, *string_name*, specifies the string to parse.

The *var_1* parameter specifies the first component of the parsed string. (A string can be parsed into a single component.)

The *seq_1* parameter specifies the sequence of the string that serves as a delimiter between *var_1* and the *var_2* parameter, which specifies the second component of the parsed string.

Each subsequent pair of *seq_n* and *var_n* parameters specify the delimiter (*seq_n*) between the preceding component and the next component (*var_n*).

The last *var_n* parameter in the command identifies the last component of the string.

Parameters

Parameter	Description
string_name	String to parse.
var_1	First component of the parsed string placed into a variable.
seq_1	Sequence of the string serving as a delimiter between *var_1* and *var_2*.
var_2	Second component of the parsed string placed into a variable.
seq_n	Sequence of the string serving as a delimiter between the preceding *var_n* and the following *var_n*.
var_n	Next component of the string after the preceding *seq_n*.

Table 14.32 **parse** Command Parameters

Examples

The following examples illustrate how **parse** can be used.

```
parse "This is my string" var1
echo "$(var1)"
This is my string

#comment: seq_1 equals " "
parse "This is a test" var1 " " var2 " " var3 " " var4
echo "$(var1)" "$(var2)" "$(var3)" "$(var4)"
Thisisatest


#comment: seq_1 equals "."
parse "KLH1A.DEV.UDM.TEST" var1 . var2 . var3 . var4
echo "$(var4)"
TEST

set string="This is a test"
parse $(string) var1 " is a" var2
echo "$(var1)" "$(var2)"
This test
```

14.35 print

Syntax

```
print msg=message
```

Description

The **print** command prints a message in the UDM Manager's transaction output.

If the message contains spaces, it should be surrounded by double (**"**) quotation marks.

Parameters

Parameter	Description
msg=*message*	Message to be printed by the UDM Manager in its transaction output.

Table 14.33 **print** Command Parameter

Examples

To print a simple message:

```
print msg="This is a simple message."
```

To print a message containing the value of a variable:

```
print msg="The current value of the UDM return code is: $(_rc)"
```

14.36 query

Syntax

```
query
```

Description

The **query** command prints the UDM Manager version.

If a session is established, the primary and secondary transfer servers display their host name and UDM version, as in this sample:

```
machine1: Connected to UDM host endymion version 1.1.0 Level 0
machine2: Connected to UDM host hyperion version 1.1.0 Level 0
```

Each line is prefixed with the logical name of a transfer server and contains its host name and version information for the UDM server instance.

14.37 quit

Syntax

```
quit
```

Description

The **`quit`** command exits the UDM Manager. If a session is open, the UDM Manager issues a **`close`** command before exiting to end the transfer session. UDM returns the value of the internal variable **`_rc`** when it exits (see set command).

Note: The UDM Manager will issue a **`quit`** command automatically if the return value of any command is greater than the value of the internal variable, **`_halton`** (if the value of **`_halton`** is greater than zero).

14.38 rename

Syntax

```
rename logical-name old-filename new-filename
```

Description

The **rename** command renames a file.

Parameters

Parameter	Description
logical-name	Logical name of the transfer server on which you want to rename a file.
old-filename	Current name (filename, path, or absolute path) for the file that you want to rename.
new-filename	New name (filename, path, or absolute path) to which the file is to be renamed.

Table 14.34 **rename** Command Parameter

14.39 replace

Syntax

```
replace variable_name oldsequence newsequence
        [num=index] [all=yes|no] [case=yes|no]
```

Description

The `replace` command replaces one or more instances of a sequence with another sequence.

The first parameter, *variable_name*, is the name of an existing variable or list element on which to perform the replace. The *oldsequence* parameter specifies the sequence to be replaced. The *newsequence* parameter specifies the sequence replacing *oldsequence*.

The optional `num` parameter specifies a one-based sequence number that identifies the occurrence of the sequence that is replaced.

The optional `all` parameter indicates whether all instances of the old sequence are replaced (**yes**) or just the first instance encountered (**no**). If `all` is not specified, it is assumed that only the first instance is replaced.

If `all` equals **yes**, and `num` also is specified, `num` is ignored.

The optional `case` parameter specifies whether the comparison to match the old sequence is case sensitive (**yes**) or case insensitive (**no**). [Default is **no**.]

Parameters

Parameter	Description
variable_name	Name of the existing variable or list element on which to perform the replace.
oldsequence	Sequence to be replaced.
newsequence	Sequence replacing *oldsequence*.
num=*index*	One-based sequence number that identifies the occurrence of *oldsequence*.
all=**yes** | **no**	Specification for whether all instances (**yes**) of the sequence are replaced or only the first instance (**no**) of the sequence.
case=**yes** | **no**	Specification for whether the comparison to match the old sequence is case sensitive (**yes**) or case insensitive (**no**). [Default is **no**.]

Table 14.35 **replace** Command Parameters

Examples

The following examples use a predefined variable, **`mystring`**, with a value of **`abcabcabc`**.

```
replace mystring ab " " all=yes
echo "$(mystring)"
 c c c

replace mystring "abc" "123" num=1
echo "$(mystring)"
123abcabc
```

14.40 report

Syntax

```
report progress=yes|no
```

Description

The **`report`** command sets reporting options for UDM.

Parameters

Parameter	Description
progress=**yes** | **no**	Turns on or off the writing of periodic transfer progress messages by the UDM Manager during a file transfer operation.

Table 14.36 **report** Command Parameters

14.41 resetattribs

Syntax

```
resetattribs logical-name
```

Description

The **`resetattribs`** command resets the attributes for all UDM file systems on the transfer server with the specified logical name.

Parameters

Parameter	Description
logical-name	Logical name of the transfer server whose attributes are to be reset to their default values.

Table 14.37 **resetattribs** Command Parameters

14.42 return

Syntax

```
return [return-value]
```

Description

The **`return`** command, generally used in conjunction with an **`if`** statement, performs the following:

1. Stops executing the current script immediately.
2. Returns execution to the calling script immediately after the **`call`** command used to invoke the current script.

If the current script was called by invoking UDM with the SCRIPT_FILE option, UDM will exit.

Optionally, the **`return`** command can specify a numeric return value. UDM will set the **`_rc`** variable (UDM's return code) to this value.

Parameters

Parameter	Description
return-value	Return value to which the **`_rc`** variable is set, specifying when to return.

Table 14.38 **return** Command Parameters

Examples

To return from the currently executing script using the current value for **`_rc`**:

```
return
```

To return from the currently executing script using a specific numeric return value to set **`_rc`** to:

```
return 4
```

14.43 reverse

Syntax

```
reverse [variable_name]
```

Description

The `reverse` command reverses the order of all characters in the string of an existing variable or element (*variable_name*).

Parameters

Parameter	Description
variable_name	String in which all characters will be reversed.

Table 14.39 **reverse** Command Parameters

14.44 savedata

Syntax

```
savedata [data_element_name=file_spec]
```

Description

The **savedata** command writes each line of a data element to a file on disk.

If the data element was created (via the data command) with a **resolve** value of **all** or **defined**, variable references within each line will be resolved, where appropriate. In such cases, if a variable reference cannot be resolved when called for, an error will be issued.

savedata issues an error if it cannot:

- Open the specified file.
- Write the entire contents of the data element to that file.
- Close the file.

It also issues an error if the named data element does not exist.

Parameters

Parameter	Description
data_element_name	Name of the data element.
file_spec	Name of the file on which to write each line of the data element.

Table 14.40 **savedata** Command Parameters

z/OS

When issuing **savedata**, *file_spec* must be a DD name in the format of dd:ddname, as is the case with the loaddata command. Also as with loaddata, **savedata** only handles files on the file system local to the manager. You cannot use **savedata** to write files on other systems.

IBM i

file_spec must specify a LIB file system file; **savedata** does not support the HFS file system.

Examples

To save a data element called **`mydata`** to a DD name under z/OS called **`export`**, execute:

```
savedata mydata=dd:export
```

To save the data element to the current directory, execute:

```
savedata mydata=mydata.txt
```

To save the data element using an absolute path, execute:

```
savedata mydata="C:\documents and settings\playground\stuff.txt"
```

14.45 set

Syntax

```
set    [built-in_name=built-in_value_1]…[built-in_name=built-in_value_n]
       [global_name=global_value_1]…[global_name=global_value_n]
```

Description

The **set** command sets values for the UDM Manager's built-in (pre-defined) variables and global variables. (Each **set** command parameter represents a variable.)

For detailed information on these variables, see Chapter 15 UDM Scripting Language.

Issuing the set Command

1. If the **set** command is issued without any parameters (variables), all of the global variables and their current values are displayed.
2. If the **set** command is issued with variable names but no following equal signs (**=**), the values to which the variables resolve are displayed.
3. If the **set** command is issued with variable names followed by an equal signs (**=**) but no values, the values are set to an empty string.

Parameters

Parameter	Description
built-in_name=*built-n_value_1*	Value of a built-in (pre-defined) variable
built-in_name=*built-n_value_n*	Value of a built-in (pre-defined) variable
global_name=*global_value_1*	Value of a global (user-defined) variable
global_name=*global_value_n*	Value of a global (user-defined) variable

Table 14.41 **set** Command Parameters

Example

To set UDM to echo lines and print line numbers on command errors:

```
set _echo=yes _lines=yes
```

14.46 status

Syntax

```
status
```

Description

The status command displays the current connection status.

If a session is open, the UDM Manager displays the following information for each server:

- Host name
- UDM version
- Logical name

If a session is not open, the UDM Manager displays a status message indicating that there currently is no session established.

Example

Sample output from the command under a three-party transfer session:

```
Three-party session established with ntmachine(1069560889) and
mvsmachine(1069560900)
```

The numbers in parentheses after each logical name are the component identifiers of the UDM server running on that machine.

14.47 strip

Syntax

```
strip variable_name sequence [num=index] [all=yes|no] [case=yes|no]
```

Description

The **strip** command strips occurrences of a sequence from a string.

The first parameter, *variable_name*, is the name of an existing variable or list element on which to perform the strip. The *sequence* parameter specifies the sequence to be stripped.

The optional **num** parameter indicates the occurrence number of the sequence that is to stripped.

The optional **all** parameter indicates whether all instances (yes) or only the first instance (no) of the sequence are stripped. (If **all** is not specified, it is assumed that only the first instance is stripped).

The optional **case** parameter specifies whether the comparison to find the sequence is case sensitive (**yes**) or case insensitive (**no**). **[Default is no.]**

Parameters

Parameter	Description
variable_name	Name of the existing variable or list element on which to perform the strip.
sequence	Sequence to be stripped
num=*index*	One-based sequence number that identifies the occurrence of the sequence to be stripped.
all=**yes** | **no**	Specification for whether all instances (**yes**) of the sequence are replaced or only the first instance (**no**) of the sequence.
case= **yes** | **no**	Specification for whether the comparison to match the old sequence is case sensitive (**yes**) or case insensitive (**no**). [Default is **no**.]

Table 14.42 **strip** Command Parameters

Examples

The following **`strip`** command examples use a predefined variable, **`mystring`**, with a value of **`abcabcabc`**.

```
strip mystring ab all=yes
echo "$(mystring)"
ccc
```

```
strip mystring abc num=1
echo "$(mystring)"
abcabc
```

14.48 substring

Syntax

```
substring   variable_name string {pos=position | startseq=sequence
            [startseqnum=number]} {length=length | endseq=sequence
            [endseqnum=number]} [case=yes|no]
```

Description

The **substring** command finds a substring in an existing string and stores it in a variable.

The first parameter, *variable_name*, specifies the name of the variable into which the substring is placed. The *string* parameter specifies the string from which the substring is taken.

The beginning of the **substring** is marked either by a position or a start sequence and optional sequence occurrence number (similar to deletestring). The end of the substring is determined by specifying the length of the substring or an ending sequence and optional sequence occurrence number, also similar to deletestring.

The **case** parameter specifies whether comparisons for start and end sequences are case sensitive (**yes**) or case insensitive (**no**). [Default is **no**.]

_lastrc.message is set to NO_MATCH if a start or end sequence was specified and could not be found. If the position or length are specified and contain invalid values, **_lastrc.message** will contain INVALID_VALUE.

Parameters

Parameter	Description
variable_name	Name of the existing variable into which the substring is placed.
string	String from which the substring is taken.
pos=position	Starting position of string to be taken (one based index)
startseq=*sequence*	Starting position of string to be taken (following a specific character sequence)
startseqnum=*number*	Occurrence number of starting position of string to be taken (following a specific character sequence)
length=*length*	Length of the string to be taken
endseq=*sequence*	Ending position of string to be taken (following a specific character sequence)
endseqnum=*number*	Occurrence number of ending position of string to be taken (following a specific character sequence
case=**yes** | **no**	Specification for whether or not the comparisons of the start and end sequences are case sensitive (**yes**) or case insensitive (**no**). [Default is **no**.]

Table 14.43 **substring** Command Parameters

Examples

The following examples illustrate the **substring** command:

```
substring res "This is fun" pos=6 length=2
echo "$(res)"
is

substring res "This is less fun" startseq=" " endseq=" "
echo "$(res)"
is

substring res "This This is fun fun" startseq=" " endseq=" " +
          endseqnum=3
echo "$(res)"
This is fun
```

14.49 truncate

Syntax

```
truncate variable_name length=length
```

Description

The **truncate** command truncates a string to a specific length.

The first parameter, *variable_name*, is the name of an existing variable or list element to truncate. The *length* parameter specifies the length to which the string is truncated.

Parameters

Parameter	Description
variable_name	Name of the existing variable or list element to truncate.
length=*length*	Length to which the string truncates.

Table 14.44 **truncate** Command Parameters

Examples

The following examples assume that a variable, **mystring**, exists with a beginning value of **12345789**.

```
truncate mystring length=4
echo "$(mystring)"
1234
```

```
truncate mystring length=8
echo "$(mystring)"
12345678
```

14.50 upper

Syntax

```
upper variable_name
```

Description

The **upper** command forces all alphabetic characters in a specified variable or list element (*variable_name*) to upper case.

Parameters

Parameter	Description
variable_name	Variable or list element in which to force alphabetic characters to upper case.

Table 14.45 **upper** Command Parameters

Example

The following example assumes that a variable, **myvar**, exists with a beginning value of **abcXYZ**:

```
set myvar=abcXYZ
upper myvar
echo $(myvar)
```

The output would be:

```
ABCXYZ
```

CHAPTER 15

UDM Scripting Language

15.1 Overview

This chapter provides information on the Universal Data Mover (UDM) scripting language.

UDM has an easy-to-learn scripting language that can be used to give instructions to UDM in both interactive and batch mode. While simple to use, UDM's scripting language has some powerful features, such as the ability to nest script file calls up to ten levels deep, and parameters.

15.2 UDM Commands

The UDM Manager processes commands using UDM's scripting language.

Table 15.1, below, identifies all of the UDM commands.

Each **Command Name** is a link to detailed information about that command.

Command Name	Description
appenddata	Appends a line of text to the end of an existing data element, or creates a new data element. containing that line of text.
attrib	Sets the file system attributes that govern the transfer operations on the host with the specified logical name.
break	Stops iterating through a **`forfiles`** loop and picks up execution at the script line immediately following the end statement marking the end of the **`forfiles`** loop.
call	Loads and executes a command script.
cd	Changes the working directory (on UNIX, Windows, IBM i, and file system HFS) or if z/OS, the current data set qualifiers (for DSN and DD file systems) on the specified logical machine to the specified path.
close	Closes the current transfer session.
closelog	Closes the open log file.
compare	Compares two strings of data.
copy	Initiates a copy operation.
copydir	Initiates a copy operation that recurses into subdirectories.
data	Defines an in-stream data element that can be passed as input for other commands.
debug	Turns debug information on and off.
delete	Deletes a file (or series of files if file-spec contains any wildcards) from the transfer server with the corresponding logical name.
deletestring	Removes a substring from an existing string.
echo	Sends text to standard out (stdout).
echolog	Sends text to an open log file.
exec	Executes system commands on remote machines.
execsap	Executes SAP events.
exit	Exits the UDM Manager (same as the **`quit`** command).
filesys	Sets the file system with which the server with the specified logical name is working.
filetype	Set a series of masks and corresponding transfer mode types.
find	Finds a specific occurrence of a substring in an existing string or list element.
format	Creates a formatted string.
insertstring	Inserts a substring into an existing string.
loaddata	Loads the contents of a data element from a file, instead of setting them in a script via the data command.

Command Name	Description
logdata	Writes the content of a data element to the open log file.
lower	Forces all alpha characters in a given variable or list element to lower case.
mode	Sets the current transfer mode.
move	Initiates a move operation.
open	Opens a UDM session.
openlog	Opens a log file on disk for writing custom log information.
pad	Takes a string in an existing variable or list element and pads it to make it the given length.
parse	Parses a string, placing the components of the string into variables.
print	Prints a message in the UDM manager's transaction output.
query	Prints out the UDM Manager version.
quit	Exits the UDM Manager (same as the **`exit`** command).
rename	Renames a file.
replace	Replaces one or more instances of a sequence with another sequence.
report	Sets UDM's reporting options.
resetattribs	Resets the attributes for all UDM file systems on the transfer server with the specified logical name.
return	Stops executing the current script immediately and returns execution to the calling script immediately after the **`call`** command used to invoke the current script.
reverse	Reverses the order of all characters in the string of a specified existing variable or element.
savedata	Writes each line of a data element to a file on disk.
set	Sets the UDM Manager's built-in and global variable values.
status	Displays the current connection status.
strip	Strips occurrences of a sequence from a string.
substring	Finds a substring in an existing string and stores it in a variable.
truncate	Truncates a string to a specific length.
upper	Forces all alpha characters in a given variable or list element to upper case.

Table 15.1 UDM Commands

15.3 UDM Command Format

All UDM commands conform to the following format:

```
command [parameter_1[=value_1]]…[parameter_n[=value_n]]
```

15.3.1 Basic Rules

The following basic rules apply to all UDM commands.

Parameters

Each command can have zero or more parameters. Each parameter can have a value, which immediately must follow an equal (**=**) sign.

Spaces

A space must precede each parameter or parameter and value.

Value names, such as a filename with a long path under Windows, can include spaces. To indicate such values, use quotation marks (**"**).

For example:

```
copy src="c:\program files\somefile.txt" dst=test.txt
```

Escape Sequences

Double Quote Marks

To include quotation marks (**"**) as part of the token, use two quotation marks in a row:

```
> echo "This word is ""quoted""!"
This word is "quoted"!
```

Other Printable Characters

When processing tokens that are inside quotation marks, all other printable characters - except variable references - are ignored as being part of the language.

If you want to assign a variable to have a value of a language symbol, such as an equal sign (**=**), you must enclose it in quotation marks:

```
> set myvar="="
> echo $(myvar)
=
```

Line Continuation

If a command is too long for a single line, it can be continued on one or more following lines by placing either of the following characters as the last character in each line break:

- Plus sign (**+**)
  Retains leading white space on the next line when assembling the finished line.
- Minus sign (**-**)
  Trims the leading white space.

For example:

```
This is +
    a test
```

Yields the following line:

```
This is     a test
```

```
This is -
    a test
```

Yields the following line:

```
This is a test
```

Comments

A script also can have comments: lines of user-specified text indicating information about the script and the operations taking place. Comment lines begin with the hash (**#**) mark. White space characters can precede the hash (**#**) mark.

15.3.2 Sample UDM Script

The following is a sample UDM script:

```
  # Open a transfer session
  open src=* dst=ntmachine

  # Copy command using line continuation
  copy src=test.txt +
  dst=test.txt

if 8 EQ $(_lastrc)
  print msg="The last command resulted in an error"
end

  # Close the transfer session and exit UDM
  quit
```

15.3.3 Expressions

The following basic rules apply to expressions in UDM commands.

Appearance

An expression can appear either as a parameter or its value. It must comprise the entire parameter or value in which it appears, not just part of it.

For example, in the following command, **`<2 + 2>`** is not an expression:

```
echo "2 + 2 = <2 + 2>"
```

It is merely part of the quoted string, and the output would be:

```
2 + 2 = <2 + 2>
```

In order to treat **`<2 + 2>`** as an expression, the command must be:

```
echo "2 + 2 = " <2 + 2>
```

The output of this command is:

```
2 + 2 = 4
```

Integer Only

Although floating point number are allowed in expressions, everything is evaluated as an integer. The only exception is that the EQ - Equal and NE - Not Equal comparators can be used to compare strings as well as numbers.

Delimiters

All expressions must be bound by left angle (**<**) and right angle (**>**) brackets.

For example:

```
set value=<2 + 2>
```

Operand / Operator Delimiters

Operands and operators in an expression must be separated by a space.

For example, the following is not legal:

```
<2+4>
```

There must be a space before and after the operator < **+** >:

```
<2 + 4>
```

Operator Precedence

The operator order of precedence (and reading left to right) is:

1. NOT
2. *, /, and %
3. + and **-**
4. EQ, NE, LT, GT, LE, and GE
5. AND, OR, and XOR

To manually indicate that an operation is of higher precedence, enclose it in parentheses: **(** and **)**. An expression is evaluated going from the inner most set of parentheses out. Sets on the same level are evaluated left to right.

The following examples illustrates how parentheses can affect results.

In the following expression, where < ***** > takes precedence over < **+** >:

```
<5 + 4 * 2>
```

The expression yields the following value:

```
13
```

However, when the expression includes the following parentheses:

```
<(5 + 4) * 2>
```

The expression yields the following value:

```
18
```

Nesting

Expressions can be nested in order to indicate desired change precedence. Nested expressions are bound by parentheses: **(** and **)**.

When nested expressions are a part of an expression, the deepest nested portions are evaluated first. Also, spaces are not required between the **(** and **)** when they are used for nesting, as they are between operators and operands.

For example:

```
> echo <5 + 5 * 2>
15

> echo <(5 + 5) * 2>
20

> echo <2 * 4 + (2 * (7 + 2))>
26

> echo <2 * 4 + 2 * 7 + 2>
24
```

Operations

All operations take the following form:

```
left-value operator right-value
```

The `left-value` and `right-value` can be:

- Strings (**`EQ`** and **`NE`** only)
- Numbers
- Variable references
- Other operations

Table 15.2 identifies and describes all of the operators for UDM command expressions.

Operator	Description
EQ	Compares the value on the left to the value on the right. If the two are equal, the result is 1, otherwise it is 0. Both the left an right values can be strings or numbers. If they are strings, the comparison is case insensitive.
NE	Works like the equal operator, except that it results in 1 if the left and right value are not equal and 0 if they are equal.
LT	Results in a value of 1 if the left value is less than the right value, otherwise it results in 0. This is a numeric operator.
GT	Results in a value of 1 if the left value is greater than the right value, otherwise it results in 0. This is a numeric operator.
LE	Results in a value of 1 if the left value is less than or equal to the right value, otherwise it results in 0. This is a numeric operator.
GE	Results in a value of 1 if the left value is greater than or equal to the right value, otherwise it results in 0. This is a numeric operator.
AND	Results in a value of 1 if both the left value and right value are not 0, otherwise it results in 0.
OR	Results in a value of 1 if either the left value or the right value are not 0, otherwise it results in 0.
XOR	Results in a value of 1 if either the left or the right values are not 0, but not both. If both the left and right values are 0, then the result of the XOR operator is 0.
NOT	Unlike all of the operators, the NOT operator has only one operand that appears to the right of the NOT operator. This operation evaluates to one if the operand is zero and zero if the operand is non-zero.
+	Result is the sum of the left and right values.
-	Result is subtracting the right value from the left value.
*	Result is the product of the left and right values.
/	Result is the left value divided by the right value. Assuming integer-only math, the remainder is discarded.
%	Result is the remainder of the left value divided by the right value.

Table 15.2 UDM Command Expressions - Operators

15.3.4 Strings in Expressions

You can use strings as operators in **EQ** and **NE** expressions. However, to avoid ambiguity, strings must be quoted explicitly (even if they are contained in a variable reference); otherwise, an error is generated.

The following expression correctly compares strings:

```
<"yes" EQ "yes">
```

Conversely, the following expression results in an error because the strings being compared are not quoted:

```
<yes EQ yes>
```

For these next two examples, assume that a variable called **myvar** has been defined with a value of *yes*.

This expression results in an error because when **myvar** is referenced, its value (*yes*) is not quoted:

```
<$(myvar) EQ "yes">
```

Instead, the correct expression should be:

```
<"$(myvar)" EQ "yes">
```

Index Position and Sequence

When working with strings, the following policy regarding the inclusion/exclusion of positional length/index and sequences should be followed:

- Positional Index or Length = Include the value at that position in the operations.
- Sequences = Exclude the value in operations.

(See deletestring, insertstring, and substring in Chapter 14 UDM Commands.)

15.3.5 Examples of Expressions

```
> set x=<2 + 4>
> set y=<$(x) * 10>
> echo "x * y = " <$(x) * $(y)>
x * y = 360

> set x=4
> set y=2
> set z=<($(x) + $(y)) * 10>
> echo "z = $(z)"
z = 60

> set x=10 z=20
> set q=<$(x) LT $(z)>
> echo "q = $(q)"
q = 1
```

15.4 Script Files

You can execute UDM commands interactively or in batch, depending on the operating system.

On all platforms other than IBM i, the batch method reads the UDM commands from a script file specified by the SCRIPT_FILE option.

z/OS

SCRIPT_FILE is not specified, the commands are read from the file allocated to the **UNVSCR** DD statement in the execution JCL.

IBM i

The script file and member can be specified by position as the first two command parameters or by using the SRCFILE and SRCMBR command parameters.

The UDM interactive method also can read UDM commands from a script file.

Syntactically, there is no differences in the command structure between the two methods, with the exception that commands in the batch method script files can contain parameters.

15.4.1 Invoking UDM in Batch Mode with Commands from a Script File

To launch UDM in batch from the Windows or UNIX command line, use the SCRIPT_FILE option to specify the filename of the script.

For example:

```
udm -s script_filename
```

Scripts also can have parameters, which are specified in the same name=value format of command parameters. To specify options parameters for a script file on the command line, use the SCRIPT_OPTIONS option.

For example:

```
udm -s copyfiles.udm -o "file=* source=c:\source dest=c:\destination"
```

In this example, UDM is invoked with a script file name **copyfiles.udm**. It has three parameters that are passed to the script file:

1. **file**, with a value of *
2. **source**, with a value of **c:\source**
3. **dest**, with a value of **c:\destination**

When UDM has finished executing the **copyfiles.udm**, it will terminate.

The following example shows the same options for a batch execution in z/OS:

```
//jobname JOB CLASS=A,MSGCLASS=X
//STEP1 EXEC UDMPRC
//UNVSCR1 DD *
udm commands??
/*
//SYSIN DD *
-s UNVSCR1 -o "file=* source=c:\source dest=c:\destination"
/*
```

For IBM i examples, see Sections Running UDM from a Script and Running UDM in Batch Mode.

15.4.2 Invoking UDM Interactively with Commands from a Script File

You also can invoke scripts directly from the UDM prompt in interactive mode (or as part of a script file itself) using the `call` command. Any parameters to the `call` command are passed on to the script being called.

The following example illustrates the `call` command executing the **`copyfiles.udm`** script (as identified in the previous example):

```
UDM
UDM>call copyfiles.udm file=* source=c:\source dest=c:\destination
UDM>
UDM>quit
```

Unlike executing a script from the command line, UDM will not exit automatically when it finishes processing a script invoked using the `call` command.

Stoneman's Tip

If you are passing a large number of parameters to a script, you may want to break up the `call` command into multiple lines.

You can do this by putting a **+** at the end of each line break, except for the last line.

However, this method cannot be used for invoking UDM script files with -s and -o command line options (SCRFILE and OPTIONS command parameters under IBM i).

15.4.3 Invoking Scripts from within Scripts

As mentioned in the previous section, scripts can use the **`call`** command to invoke other scripts. Scripts can be nested up to ten levels deep. As each script finishes processing, control is returned to the script that invoked it immediately following the point of invocation.

When nesting scripts, parameters from higher-level invocations are available to scripts invoked at a lower level. If the same parameter name is used in more than one invocation of a series of nested scripts, the value for the instance of the parameter at the lowest level is used.

15.4.4 Parameter Processing

A parameter is referenced inside of a UDM script using the following format:

```
$(parameter_name)
```

When a parameter reference is encountered, it is replaced with the value of parameter matching the enclosed name. Continuing with **`copyfiles.udm`** script example (used previously in this Section, 15.4 Script Files), a reference to **`$(source)`** would be replaced with **`c:\source`**.

An example of how the **`copyfiles.udm`** script might look is as follows:

```
cd src=$(source)
cd dst=$(dest)
copy src=$(file)
```

In this example:

1. Transfer server with the logical name **`src`** would change its directory to **`c:\source`**.
2. Transfer server with the logical name **`dst`** would change its directory to **`c:\destination`**.
3. All files in the **`c:\source`** directory then would be copied from the first transfer server over to the second.

15.5 Subroutines

UDM's scripting language provides support for subroutines.

Subroutines are portions of the script code that can be called, by name, at any point. This provides a convenient way to reuse common script code.

15.5.1 Usage

There are two parts to a subroutine:

1.	Definition	Names the subroutine and defines the script code that becomes associated with that subroutine name.
2.	Invocation	Carries out the work of lines of script associated with a subroutine.

Defining a Subroutine

```
subroutine name
[script line 1]
…
[script line n]
endsub
```

Invoking a Subroutine

```
callsub name
```

Sequence of Defining / Invoking a Subroutine

A subroutine must be physically defined *before* the **`callsub`** to the routine is used.

For example, the following subroutine will function correctly:

```
subroutine test
   echo "This is subroutine test"
   echo "$(_halton)"
endsub

echo "This is main()"
callsub test
```

However, this subroutine will fail:

```
echo "This is main()"
callsub test

subroutine test
   echo "This is subroutine test"
   echo "$(_halton)"
endsub
```

Nesting / Recursion of Subroutines

UDM allows subroutine nesting (one subroutine calls another subroutine) and recursion (a subroutine calls itself).

For example, the following illustrates subroutine nesting:

```
subroutine a
    echo "Beginning subroutine A"
    callsub b
    echo "Ending subroutine A"
endsub

subroutine b
    echo "Beginning subroutine B"
    echo "Ending subroutine B"
endsub

callsub a
```

15.5.2 Example

```
subroutine loop_increment
        echo "inside loop_increment: $(LOOP)"
        set LOOP=<$(LOOP) + 1>
endsub


echo "Starting Loop:"
set LOOP=0
if <$(LOOP) LT 1>
        callsub loop_increment
end
if <$(LOOP) LT 2>
        callsub loop_increment
end
if <$(LOOP) EQ 2>
        callsub loop_increment
end
echo "Final Value of LOOP: $(LOOP)"
```

Output

```
Starting Loop:
inside loop_increment: 0
inside loop_increment: 1
inside loop_increment: 2
Final Value of LOOP: 3
```

15.6 UDM Variables

Variables are integral data storage objects in the Universal Scripting Engine.

This section provides information on:

- Variable types
- Variable scope
- Variable reference
- Variable attributes

15.6.1 Variable Types

There are two types of variables:

1. Script
2. Global (user-defined and built-in)

Variable Names

There are no restrictions on variable names except:

- They cannot contain double-quote marks (**"**) or spaces.
- UDM reserves variable names beginning with an underscore (**_**) for its own internal (built-in) variables (see Section 15.6.8 Built-in Variables). You cannot create a script variable or user-defined global variable that begins with an underscore (**_**).

15.6.2 Variable Reference

To obtain the value of a variable, you must create a reference for that variable. The reference can appear anywhere in a script line. It is replaced with the value of the referenced variable.

Referencing the value of a global variable in a script is done exactly the same way as for a script variable:

```
$(variable_name)
```

For example:

```
set srcfile=myfile.txt
set dstfile=yourfile.txt
copy src=$(srcfile) dst=$(dstfile)
```

15.6.3 Script Variables

Script variables are user-defined variables visible only to a called script, and any of its children, for which they have been defined. When the called script has ended, the script variables' definitions are removed from the scripting engine environment.

You must use the **`call`** command to define script variables. The variables are specified as parameters in a **`call`** command, after the script name, that loads and executes a script. They are created during execution of the script.

The value of a script variable cannot be changed once it has been created.

(For detailed information on defining script variables, see Section 15.4 Script Files.)

15.6.4 Global Variables

Global variables are variables that are visible at all script levels.

There are two types of global variables:

1. User-defined (see Section 15.6.6 User-Defined Variables)
2. Built-in (see Section 15.6.8 Built-in Variables)

Global variables are permanent in scope and, once defined, last until the UDM Manager is terminated. Once defined, a global variable cannot be undefined, but its value can be changed by issuing another **`set`** command.

Issuing the **`set`** command by itself with no arguments displays all user-defined and built-in variables.

Each global variable includes the following information:

1. Name: identifies a variable as either a UDM pre-defined (built-in) variable (see 15.6.8 Built-in Variables) or a user-defined variable. A variable is referenced in a script by this name.
2. Attribute: optional entry that can be included in a variable reference. It provides additional information about a variable (see 15.6.7 Variable Attributes).
3. Value: assigned by the user and/or provided by UDM, depending on the variable. Built-in variables have pre-defined and/or user-defined values. All user-defined variables have user-defined values.

15.6.5 Scope of Script and Global Variables

A variable's scope is its visibility throughout the scripting environment.

Script variables supersede global variables in precedence. That is, if a script variable has the same name as an existing global variable, any references to that variable name will result in the script variable value, not the global variable value.

If more than one script variable exists with the same name, the script variable defined in the last **`call`** command has precedence.

For example, if UDM encounters the **`$(variable_name)`** sequence, it first checks to see if a variable with a matching name was passed into the script it currently is executing. If so, UDM uses this variable's value. If not, UDM goes up the chain of calling scripts and uses the value of the first instance of a variable that it finds that matches the name in the sequence.

If no variable with a matching name was passed into any script along the chain, UDM looks to see if a global variable exists with the name. If one is found, its value is used. If no instances are found anywhere, an error is issued.

Variable Scope Scripts

The following three scripts demonstrate variable scope:

script1.udm

```
set var1="a global variable"
set var2="a global variable"
set var3="a global variable"
print msg="The value of var1 is $(var1)"
print msg="The value of var2 is $(var2)"
print msg="The value of var3 is $(var3)"
call script2.udm var1="passed into script2"
             var2="passed into script2"
```

script2.udm

```
print msg="The value of var1 is $(var1)"
print msg="The value of var2 is $(var2)"
print msg="The value of var3 is $(var3)"
call script3.udm var1="passed into script3"
```

script3.udm

```
print msg="The value of var1 is $(var1)"
print msg="The value of var2 is $(var2)"
print msg="The value of var3 is $(var3)"
```

Running UDM and calling **`script1.udm`** produces the following results:

```
Processing script: script1.udm
The value of var1 is a global variable
The value of var2 is a global variable
The value of var3 is a global variable
Processing script: script2.udm
The value of var1 is passed into script2
The value of var2 is passed into script2
The value of var3 is a global variable
Processing script: script3.udm
The value of var1 is passed into script3
The value of var2 is passed into script2
The value of var3 is a global variable
Finished processing script: script3.udm
Finished processing script: script2.udm
Finished processing script: script1.udm
```

15.6.6 User-Defined Variables

User-defined variables are defined using the **set** command. They can have any name — except that they cannot begin with an underscore (_) character — and any value.

A user-defined variable can be called within any script or in an interactive session:

```
set variable_name=variable_value
```

The following example creates a user-defined variable called **test**:

```
set test="This is a test."
```

Multiple user-defined variables can be set with a single call to the **set** command, listing each variable's name / value pair in succession, separated by spaces:

```
set varname1=value1 varname2=value2 varname3=value3
```

To assign a value to a variable that has one or more spaces in it, the value must be quoted:

```
set lonvar="This variable has a rather long value."
```

15.6.7 Variable Attributes

In addition to accessing the value of a variable, you can access information about that variable through its attributes.

A variable attribute is referenced by putting a dot (**.**) after the variable name in a variable reference. For example:

```
$(name.attribute)
```

There are two variable attributes that can be used for any variable:

1. **`exists`**
2. **`length`**

Note: Some built-in variables have attributes specific to those variables.

exists Attribute

The **`exists`** attribute expands to **yes** if a variable with that name exists at any scope; it expands to **no** if no variable with that name exists.

Note: Unlike when referencing a variable's value, an error is not issued if the **`exists`** attribute is used for a variable name that does not exist.

Stoneman's Tip

You can use the **`exists`** attribute
in combination with the **`if`** statement
to determine if a variable exists
(and take the appropriate action if it does):

```
if $(filename.exists) EQ yes
   copy src=$(filename)
end
```

length Attribute

The **`length`** attribute expands to the length of the variable's value. If the **`length`** attribute is used for a variable that does not exist, an error is issued.

Stoneman's Tip

You can use the **`length`** attribute
in combination with the **`if`** statement
to decide if a file name is too long to copy to a remote system:

```
if $(filename.length) LE 8
   copy src=$(filename)
end
```

15.6.8 Built-in Variables

UDM provides built-in variables that are used to make available some of its internal values to UDM commands. Depending on the variable, their values are provided by UDM and/or defined via the set command.

Note: All built-in variables are preceded by an underscore (_) to indicate that they are built-in variables reserved by UDM and so do not conflict with the built-in variables and variables created in a script.

Stoneman's Tip

Version 1.1.0 of UDM had four built-in variables:
echo, **halton**, **lines**, and **rc**.

The names of these variables were not preceded with an _
as they are in version 3.1.0 and later.

For the purpose of backward compatibility,
these variables can be referenced by their 1.1.0 names as well.

Table 15.3 lists all of the UDM built-in variables and provides a link to detailed information about them in this section.

Variable	Description	Page
_date	Displays the current date in the format appropriate for the system's locale.	355
_echo	Specification for whether or not a command is echoes prior to processing.	355
_execrc	Holds the value of the process executed by the last exec command issued.	355
_file	Name of the file for the current iteration in a **forfiles** loop.	356
_halton	Return code value that causes UDM to terminate if it is greater than 0 and is equalled or exceeded by the return code value in the **_rc** variable.	357
_keepalive	Interval at which keepalive messages are sent form the UDM Manager to transfer servers.	357
_lastmsg	Contains all of the messages written in the transaction log for the last network- or file-oriented command issued.	357
_lastrc	Holds the return code of the last command issued and, optionally, an indication of what happened with the last executed statement.	358
_lines	Specification for whether or not the line number is printed with the error if a command cannot be parsed or is malformed.	359
_path	Absolute path of the file for the current iteration in a **forfiles** loop.	359
_rc	Current UDM return code.	359
_time	Current time.	360
_uuid	Generates a UUID.	360

Table 15.3 Built-In Variables

_date

The **_date** built-in variable displays the current date in the format appropriate for the system's locale.

_date has several additional variable attributes:

- **day** resolves to the day of the week.
- **month** resolves to the current month.
- **dd** resolves to a two-digit day of the month.
- **ddd** resolves to the Julian day.
- **mm** resolves to the two-digit month of the year.
- **yy** prints the two-digit year.
- **ww** resolves to the two-digit current week of the year.
- **yyyy** resolves to the four-digit year.

Note: The value of **ww** is zero-based, not one based. That is, the first week of the year is 0, the second week is 1, the third week is 2, and so on.

The **_date** variable can be referenced only in your scripts and cannot be set.

Stoneman's Tip

You can use the **_date** variable
in combination with the **print** command
to display custom date information in UDM's transaction log:

```
print msg="Today is $(_date.day), $(_date.month) $(_date.dd)"
```

Produces the following output:

```
Today is Wednesday, January 19
```

_echo

The **_echo** built-in variable specifies whether or not a command is echoed prior to processing. It can have a value of either **yes** or **no**:

- If the value is **yes**, each UDM command is echoed prior to processing.
- If the value is **no**, the command is not echoed.

The value of **_echo** can be set using the **set** command, as in the following example:

```
set _echo=yes
```

_execrc

The **_execrc** built-in variable holds the value of the process executed by the last exec command issued. The return code of the exec command itself is stored in _lastrc (and _rc, if the return code is greater than the current value of _rc).

The difference between the two values is that the return code for exec (in _lastrc and _rc) tells you whether exec was successful in executing the command that it was supposed to execute, while **_execrc** is the return code of that executed command.

The value of **_execrc** can be set using the set command.

_file

The **_file** built-in variable contains the name of the file for the current iteration in a **forfiles** loop. **_file** also has special attributes, as shown in Table 15.4, below.

For information on using **_file** and its special attributes, see _file Variable Attributes in Section 15.10.1 forfiles Built-In Variables.

Attribute Name	Description
accessdate	Date on which the file was last accessed. Format (ISO 8601) is yyyy-mm-dd.
accesstime	Time when the file was last accessed. Format (ISO 860) is hh:mm:ss.
accesstimestamp	Combination of **accessdate** and **accesstime**: yyyy-mm-dd hh:mm:ss. If the file does not have an access time, but does have a access date, 00:00:00 is used for the time portion.
createdate	Date on which the file was created. Format (ISO 8601) format of yyyy-mm-dd.
createtime	Time when the file was created. Format (ISO 8601) is hh:mm:ss.
createtimestamp	Combination of **createdate** and **createtime**: yyyy-mm-dd hh:mm:ss. If the file does not have a creation time, but does have a creation date, 00:00:00 is used for the time portion.
moddate	Date on which the file was last modified (referenced for z/OS). Format (ISO 8601) is yyyy-mm-dd.
modtime	Time when the file was last modified (referenced for z/OS). Format (ISO 8601) is hh:mm:ss.
modtimestamp	Combination of **moddate** and **modtime**: yyyy-mm-dd hh:mm:ss. If the file does not have a modification time, but does have a modification date, 00:00:00 is used for the time portion.
name	Name of the file (same as referencing **_file** itself without any attributes).
size	Size of the file (in bytes).
type	Type of file. Values are: • file • directory (also used for PDSs under z/OS) • unknown **type** has meaning in a **forfiles** statement under IBM i in the LIB file system: • If the value of _file.type is directory, the file type is a Physical file. • If the value of _file.type is file, the file type is a Save file.

Table 15.4 _file Built-in Variable – Special Attributes

_file cannot be set using the **set** command.

_halton

The **_halton** built-in variable specifies a return code value that causes UDM to terminate if that value is:

- Greater than 0
- Equaled or exceeded by the return code value in the **_rc** variable

Note: If the **_halton** value is 0, and the return code in **_rc** is 0, UDM will not terminate.

Each UDM command has a return code indicating its level of success or failure:

• 0 / none	Success or no error
• 4 / warn	Warning has been issued
• 8 / error	Error has occurred
• 16 / fatal	Fatal error has occurred

The value of **_halton** can be set using the **set** command. You also can use the convenience values of none, warn, error, and fatal (indicating 0, 4, 8, and 16, respectively) to set the value of **_halton**:

```
set _halton=error
```

_keepalive

When a UDM session is established, the UDM Manager periodically sends a keep-alive message to the transfer servers – to which the transfer servers respond – in order to make sure the session is still established.

The **_keepalive** built-in variable contains the interval (in seconds) at which these messages are sent. If it has a value of 0, no keep-alive messages are sent.

You can change this interval by setting the **_keepalive** variable using the **set** command before a session is established:

```
set _keepalive=60
```

_lastmsg

The **_lastmsg** built-in variable is a data element (that is, a simple array) that contains all of the messages written in the transaction log for the last network- or file-oriented command that was issued (open, close, attrib, cd, copy, copydir, delete, or rename).

Whenever a new network- or file-oriented command is issued, the contents of **_lastmsg** is cleared before the command is processed so that **_lastmsg** will contain only messages relating to that command.

If UDM encounters a print command while processing a network- or file-oriented command, the value of the **msg** parameter in that command is appended to **_lastmsg** as a new line.

The contents of **_lastmsg** can be listed at any time by issuing the following command:

```
data print=_lastmsg
```

_lastrc

The **`_lastrc`** built-in variable holds the return code of the last command issued.

`_lastrc` also has two special attributes: **`message`** and **`result`**.

- **`message`** contains a human-readable string indicating what happened with the last executed statement.
  - If a command could not be executed or had improper values, the value of **`_lastrc.message`** is ERROR.
  - If a command successfully executed, the value of **`_lastrc.message`** is either SUCCESS or some other message (depending upon the command).
- **`result`** holds an integer value that indicates the result of the last command executed.

  The meaning of this value depends on the command. Unless otherwise stated:
  - -1 indicates failure.
  - 0 or a positive value indicates success.

  This value does not affect the value of **`_rc`** and **`_lastrc`**.

`_lastrc` cannot be set using the **`set`** command.

Stoneman's Tip

You can use the **`_lastrc`** variable
in combination with the **`if`** statement
to take action based on the return value of the previously issued command:

```
copy src=myfile if $(_lastrc) EQ 0
   delete src=myfile
end
```

_lines

The **`_lines`** built-in variable specifies whether or not the line number of a command (relative to the script in which it occurred) is printed with the error if the command cannot be parsed or is malformed. It has a value of **yes** or **no**.

- If the value is **yes**, the line number is printed.
- If the value is **no**, the line number is not printed.

`_lines` can be set using the **`set`** command:

```
set _lines=yes
```

_path

The **`_path`** built-in variable contains the absolute path of the file for the current iteration in a **`forfiles`** loop (see Section 15.10 forfiles Statement).

`-path` cannot be set using the **`set`** command.

_rc

The **`_rc`** built-in variable holds the current UDM return code, a numeric value that indicates the highest return code received from processing all UDM commands up to that point. The value of **`_rc`** is the return code that the UDM Manager returns when it exits.

As with the **`_halton`** variable, **`_rc`** can be set, via the **`set`** command, to either of the following integers or convenience values:

• 0 / none	Success or no error
• 4 / warn	Warning has been issued
• 8 / error	Error has occurred
• 16 / fatal error	Fatal error has occurred

For example:

```
set _rc=warn
```

_time

The **_time** built-in variable displays the current time.

It has several variable attributes:

- **hh** resolves to the two-digit hour (24-hour time).
- **mm** resolves to the two-digit minute.
- **ss** the number of seconds that have elapsed since the current minute.
- **hs** resolves to the number of hundredths of a second that have elapsed since the last second.

You only can reference **_time** in your scripts; it cannot be set using the **set** command.

Stoneman's Tip

You can use the **_time** variable
in combination with the **print** command
to display custom time information in UDM's transaction log:

```
print msg="It is now $(_time.hh):$(_time.mm)"
```

Produces the following output:

```
Today is now 23:31
```

_uuid

The **_uuid** built-in variable, when referenced, generates a UUID.

For example:

```
echo $(_uuid)
1732fd12-7b07-4791-a28a-4cf0776db4f7
```

15.6.9 Logical Name Built-In Variables

When a session is established, built-in variables are created for each transfer server and contain information about each server.

The names of these variables are based on the logical name of the transfer server, preceded by an underscore. If the primary transfer server is not specified (implying a two-party transfer session), its built-in variable will have the name **`_local`**.

These logical name built-in variables persist only for the duration of the session.

They have three attributes:

1. **`host`**: contains the host name of the transfer server.
2. **`port`**: holds the port used to connect to the transfer server over.
3. **`user`**: contains the userid used to sign into the transfer server.

Examples

The following shows how these built-in variables can be used:

```
open remote=mymachine port=10000 user=me pwd=mypwd
if $(_lastrc EQ 0)
   print msg="Connected to $(_remote.host):$(_remote.port)"
   print msg="    as $(_remote.user) from $(_local)"
end
```

This example produces the following output:

```
Connected to mymachine:10000
   as me from (local)
```

15.7 if Statement

The `if` statement is used to add conditional branching of UDM commands.

An `if` statement consists of:

- Comparison operation.
- Series of UDM commands that are carried out if the comparison operation evaluates to true.
- **end** statement that indicates the end of the `if` statement.

For example:

```
if comparison
     …
     UDM commands
     …
end
```

If the comparison does not evaluate to true, UDM will pick up execution from the line after the **end** statement.

Note: The indentation of commands underneath the conditionals is not required in UDM. This is done for the sake of readability; you can indent lines in your own scripts if and as you see fit.

15.7.1 Comparison Operations

In an `if` statement, a comparison consists of three parts:

1. Left-hand value
2. Comparator
3. Right-hand value

The left-hand and right-hand values can be either:

- Variable reference
- Variable attribute
- Constant

Comparators

A comparator determines the type of comparison to be made between the left-hand and right-hand values.

There are six comparators: EQ, NE, LT, GT, LE GE.

EQ - Equal

The equal comparator, EQ, evaluates to true if both the left and right-hand values are equal to each other. If one or more of the values contains alpha characters (non-numeric), the comparison is case insensitive. That is, a word that is all lower case would be equal to the same word if it were all upper case (for example: **dog** would be equal to **DOG**).

Here are some examples of some `if` statements using the equal comparator:

```
if $(filename) EQ myfile.txt
  print msg="The name of the file is myfile.txt"
end


if 8 EQ $(_lastrc)
  print msg="The last command resulted in an error"
end


if $(filename.exists) EQ yes
  print msg="The filename variable exists"
end
```

NE - Not Equal

The not-equal comparator, NE, evaluates to true if the left-hand value is not the same as the right-hand value. As with the equal comparator, alpha character comparisons are case insensitive.

The following are some examples of the not-equal comparator:

```
if "C:\Program Files\Universal" NE $(mydir)
  print msg="This is not the Stonebranch application directory"
end


if 8 NE 0
  print msg="This will always print as 8 is not equal to 0"
end


if $(filename.exists) NE no
  print msg="The filename variable exists"
end
```

LT - Less Than

The less than comparator, LT, evaluates to true if the left-hand value is less than the right-hand value. The less than comparator performs a numeric comparison.

The following are examples of the less than comparator:

```
if 0 LT 8
   print msg="0 is less than 8"
end

if $(_rc) LT 8
   print msg="No errors have occurred"
end

if $(filename.length) LT 8
   print msg="The length of the filename is less than 8"
end
```

GT - Greater Than

The greater than comparator, GT, evaluates to true if the left-hand value is greater than the right-hand value. As with the less than comparator, the comparison is between to numeric values as in this example:

```
if 8 GT 0
   print msg="8 is greater than 0"
end
```

LE - Less Than or Equal

The less than or equal comparator, LE, is similar to the less than comparator, except that it evaluates to true if the left-hand value is less than or equal to the right-hand value as in these examples:

```
if 8 LE 8
   print msg="8 is less than or equal to 8"
end

if $(filename.length) LE 8
   print msg="The length of the filename is less than or equal to 8"
end
```

GE - Greater Than or Equal

The greater than or equal comparator, GE, is similar to the greater than comparator, except that it evaluates to true if the left-hand value is greater than or equal to the right-hand value as in this example:

```
if $(filename.length) GE 9
  print msg="The filename is longer than 8 characters"
end
```

15.7.2 Adding an Alternate Path with else Statement

Alternate Path without else Statement

Often there are occasions where you may want to take one branch if some condition is true and another branch if that condition is false, instead of merely picking up execution after the **end** statement. (Those lines would be executed if the condition was true as well, only after executing the statements inside the **if-end** pair.)

This could be accomplished by two well-phrased **if** statements, one following the other, as in this example:

```
if <$(_rc) GE 8>
  echo "There has been an error"
end
if <$(_rc) LT 8>
  echo "There has not been an error"
end
```

Flaws in this Methodology

However, while this is a perfectly valid method, it suffers from two potential flaws:

First and foremost, people may find such logic difficult to read, thus making your UDM scripts more difficult to maintain, especially by those who had not written them in the first place.

Second, if the comparison operation contains a variable and evaluates to true for the first comparison, it is possible something occurs in the statements inside the **if-end** pair that changes the value of the variable and makes the second comparison evaluate to true as well.

For example:

```
if <$(_lastrc) GE 8>
    echo "The last command was not successful"
end
if <$(_lastrc) LT 8>
   echo "The last command was successful"
end
```

In this example, if the command executed before the first **if** statement resulted in an error, the output would have been as follows:

```
Last command was not successful
Last command was successful
```

This is because the **_lastrc** variable holds the value of the last command executed by UDM. In the example given, the command executed before the first **if** statement resulted in an error (for example, result code = 8) and would result in the first **if** statement evaluating to true.

However, the successful execution of the `print` command inside the first `if` statement would result in `_lastrc` being set to 0, which would in turn mean the second `if` statement would evaluate to true, thus printing the second message. This would not have been what was intended.

In this contrived example, it is rather easy to see what went wrong and come up with a workaround: in this case, creating a new global variable into which to save the value of `_lastrc` - for example: `set newvar=$(_lastrc)` - and using the new variable in the comparison operations instead of `_lastrc` as its value would not be overwritten. For longer and more complex scripts, however, this may not be the case.

Alternate Path with else Statement

UDM offers an easy solution with the `else` statement. As part of the `if` statement, the `else` statement can be used to provide an alternative path to take if the comparison evaluates to false.

The general format of an `if` statement when an `else` statement is used with it is:

```
if expression
…
[else
…]
end
```

In this `if` statement, the parameter for the statement is an expression. If the expression evaluates to a value that is not equal to zero, the positive branch is taken; otherwise the negative (`else`) branch is taken if one exists.

Examples

```
if <$(_rc) EQ 0>
    echo "Everything worked okay"
else
    echo "Something went wrong"
end


if ("$(myvar.exists)" EQ "yes">
    echo "The variable, myvar, has been defined."
end
```

Note: The previous style of UDM `if` statements, shown in the following example, still is valid:

```
if <$(_lastrc) GE 8>
  print msg="The last command was not successful"
else
  print msg="The last command was successful"
end
```

15.7.3 Nested Conditionals

For complex and powerful operations, `if` statements can be nested inside of each other.

For example:

```
copy src=$(filename)

if $(_lastrc) EQ 0
   delete src=$(filename)

   if $(_lastrc) NE 0
      print msg="The source file could not be deleted."
   end

else
   print msg="The copy operation failed."
end

print msg="The operation completed with a return code of $(_rc)."
```

Stoneman's Tip

Indenting lines underneath conditionals by putting spaces at the front of them, although not necessary, provides a visual cue that those lines are to be executed due to the evaluation of a conditional.
Using this technique with nested conditionals provides an easy way to tell at which 'level' each of the commands belong.

In addition, leaving a blank line before and after a conditional (not required by UDM) provides a way to visually indicate a block of related script commands.
This improves the readability and maintainability of scripts in the future.

15.7.4 Returning Early Using the return Command

At times, it is useful to be able to exit from processing a single script file in the middle of that script file if certain processing conditions are not correct. For this, UDM provides the **`return`** command, which takes the following format:

```
return [value]
```

The **`return`** command stops processing of the current script and returns control to the calling script at the point immediately following the script call (just as if the script had executed completely without calling the **`return`** command). If there was not a calling script, and UDM is not running interactively, UDM will exit. The **`return`** command also can be followed by an optional value. If this is the case, UDM's return code (held by the **`_rc`** built-in variable) is set to this value upon executing the **`return`** command.

Stoneman's Tip

One common use of the **`return`** command is to exit from a script if the previous operation failed. (The **`_halton`** variable can be used for this situation if you want to exit from UDM altogether.)

However, if you only want to exit the current script, you can couple the **`return`** command with an **`if`** statement and the **`_lastrc`** built-in variable:

```
if $(_lastrc) NE 0
   return $(_lastrc)
end
```

15.8 while Statement

The `while` statement implements a simple `while` loop.

The syntax of the `while` statement is:

```
while expression
...
end
```

In this case, the loop iterates (executing the commands between the `while` and `end` statements) as long as the expression evaluates to a value that is not zero.

If the expression evaluates to a value of zero, code execution picks up at the point immediately following the end of the `while` loop.

For example:

```
set n=1
while <$(n) LE 10>
    echo $(n)
    set n=<$(n) + 1>
end
```

15.9 fordata Statement

The **`fordata`** statement iterates through a data element, once for each line. For each iteration, a variable provided by the user is set to hold the contents of the line in the data element corresponding to the current iteration.

The syntax of the **`fordata`** statement is:

```
fordata variable-name=data-element
...
end
```

Example

```
set i=1
loaddata mydata=mydata.txt
fordata line=mydata
    echo "$(i): $(line)"

    compare "$(line)" "exit" case=yes

    if <"$(_lastrc.message)" EQ "MATCH">
        echo
        echo "Data contains an 'exit' command"
        echo
    end

    set i=<$(i) + 1>
end
```

If a data element called **`mydata.txt`** contained the following contents:

```
cd /
ls -al
exit
```

Running this script against the contents of **`mydata`** would produce the following results:

```
1: cd /
2: ls -al
3: exit
Data contains an 'exit' command
```

15.10 forfiles Statement

UDM provides a powerful iterative loop structure, **`forfiles`**, that iterates through a series of statements for each file found that matches a file specification.

The syntax of the **`forfiles`** statement is:

```
forfiles logical_name=file_spec
         [sortby=attribute-name[,ascending | descending]]
   …
   UDM commands
   …
end
```

`logical_name` is the logical name of a transfer server.

`file_spec` is the file specification used to select files for the iteration (see Section 15.10.2 forfiles File Specification).

From the specified transfer server, UDM builds a list of files that match the file specification. UDM then executes all of the commands listed between the **`forfiles`** statement and the **`end`** statement, once for each file in the list.

The optional **`sortby`** parameter specifies the name of a special attribute ***`attribute-name`*** of the _file built-in variable (see Table 15.4 _file Built-in Variable – Special Attributes). The list of files that match ***`file_spec`*** will be sorted based on the value of ***`attribute-name`***. **`ascending`** and **`descending`** specify whether the matching files are listed in ascending or descending order. (If neither is specified, the list is sorted in ascending order.)

Since having a **`sortby`** attribute in the **`forfiles`** loop implies that the file attributes will be used, the file attributes will be retrieved regardless of whether or not **`fileattrib=yes`** is present.

Examples

To obtain a file list, sorted by creation date (earliest to latest):

```
forfiles src=*.txt sortby=createdate
    # Do some stuff
end
```

Top obtain a file list, ordered by file size from largest to smallest:

```
forfiles src=*.exe sortby=size,descending
    # Do some more stuff
end
```

An error would be produced if **`sortby`** was present without a value or if it referred to an attribute that does not exist for the _file variable.

15.10.1 forfiles Built-In Variables

The **forfiles** statement utilizes two built-in variables: _file and _path. These variables are set by UDM to contain the file name and absolute path of each file in the list for each iteration.

For an example, assume the following:

- Windows machine with logical name **nt**.
- Directory **C:\Example** on the Windows machine.
- Three files – **file1.txt**, **file2.txt**, and **file3.txt** – in the directory.

The following script segment prints the file name and absolute path of each file in the directory:

```
forfiles nt=C:\Example\*
   print msg="Filename: $(_file)   Abs. Path: $(_path)"
end
```

Executing this would build a file list containing the files: **file1.txt**, **file2.txt**, and **file3.txt**. Since there are three files in the list that was built, UDM would iterate through the loop three times:

1. During the first iteration through the loop, the **_file** variable would contain **file1.txt** and the **_path** variable would contain **C:\Example\file1.txt**.
2. During the second iteration, the **_file** variable would contain **file2.txt** and the **_path** variable would contain **C:\Example\file2.txt**.
3. During the third and final iteration, the **_file** variable would contain **file3.txt** and the **_path** variable would contain **C:\Example\file3.txt**.

This script segment would result in the following output:

```
Filename: file1.txt   Abs. Path: C:\Example\file1.txt
Filename: file2.txt   Abs. Path: C:\Example\file2.txt
Filename: file3.txt   Abs. Path: C:\Example\file3.txt
```

_file Variable Attributes

The _file variable also has special attributes that further define a file (see Table 15.4 _file Built-in Variable – Special Attributes).

For efficiency reasons, all of these attributes – other than **name** and **type** – are retrieved only as requested. You can request to retrieve the file attributes by adding **fileattrib=yes** to the end of the **forfiles** call.

For example:

```
forfiles src=*.txt fileattrib=yes
    echo "$(_file) is $(_file.size) bytes in size."
end
```

If the information for an attribute cannot be obtained, its value is set to an empty string.

z/OS

z/OS datasets store only **createdate** and **accessdate**. There is no time (**createtime** and **accesstime**) associated with these dates, nor do z/OS datasets store **moddate** or **modtime**.

15.10.2 forfiles File Specification

The file specification portion of the **forfiles** statement, **file_spec**, tells UDM how to build its list of files. It takes the same format as the copy command file specification and can contain wildcards.

To list all of the members in a PDS on a z/OS system, you can issue the following commands:

```
forfiles zos=MYHLQ.MYPDS(*)
   print msg=$(_file)
end
```

This would print the name of each member in a PDS called **MYHLQ.MYPDS**.

For Windows, UNIX, and IBM i systems (as well as the HFS file system under z/OS), you can list all of the files in the current directory with the following UDM commands:

```
forfiles local=*
   print msg=$(_file)
end
```

To find all of the files ending in **.txt** in a particular directory, **mydir** in this example, issue the following **forfiles** statement:

```
forfiles local=mydir/*.txt
   print msg=$(_file)
end
```

A question mark (**?**) can be used as a wildcard for a single character. In the previous example, lets assume **mydir** contains the files: **file**, **file1**, **file2**, and **file3.txt**.

Executing the following:

```
forfiles local=mydir/file?
   print msg=$(_file)
end
```

Will result in the following output:

```
file1
file2
```

The **forfiles** file specification can contain wildcards for any UDM files system.

Under z/OS, however, the wildcards only can be used to reference a member of a PDS or PDS/E.

Stoneman's Tip

15.10.3 Breaking Out Using the break Command

The break command is a powerful command that can be issued from inside of a **`forfiles`** loop. It causes UDM to stop iterating through the **`forfiles`** loop and resume execution at the command immediately following the **`end`** statement marking the end of the loop.

Stoneman's Tip

One use for the **`forfiles`** statement is to try and copy a series of files, deleting the source file if the copy operation was successful.

The following is a sample that accomplishes this task, exiting from the loop if a file cannot be copied or if, after copying a file, it cannot be deleted:

```
forfiles local=*
    copy local=$(_file)
    if $(_lastrc) NE 0
        print msg="Could not copy $(_path)"
        break
    end
delete local=$(_file)
    if $(_lastrc) NE 0
        print msg="Could not delete $(_path)"
        break
    end
end
```

15.11 Creating In-Stream Data with the data Command

The data command can be used to define in-stream data elements that can be passed as input for other commands, such as the exec command.

The syntax for the data command is as follows:

```
data [NAME|print=NAME] [resolve=all|defined|no] [end=ENDSEQUENCE]
[DATA]
end|ENDSEQUENCE
```

15.11.1 Creating an In-Stream Data Element

An in-stream data element has four parts:

1. Name
2. Optional variable resolution method
3. In-stream data itself
4. End-of-data marker or end sequence

The name uniquely identifies the data element and is used to refer to the data element.

The optional variable resolution method tells UDM whether to resolve variables wrapped in the **$()** sequence when the data element is referred to.

- If the resolution method is **all** (default), all variables are resolved and an error is issued if the variable is not defined in UDM.
- If the resolution method is **defined**, only references to variables defined in UDM are resolved and all other **$()** references are left as is in the data element.
- If the resolution method is **no**, UDM does not try to resolve any variable references in the data when the data element is used.

The data portion of the data element is the actual data that will be used by the command that is referencing that data element.

Note: The data is used as entered, including any leading spaces or tabs; no trimming is done.

The end-of-data marker or end sequence marks the end of the data. By default, this is simply the word **end**. It must appear separately, on its own line. However, it is possible that **end** is valid instream data and you can change the end sequence with the **end** parameter of the data command.

Example

The following example shows how to use the data command in conjunction with the exec command to look through a series of copied files and display lines with the occurrence of some string under UNIX:

```
open remote=yourmachine user=someguy pwd=somepwd

data mydata resolve=all
grep "this is my sequence" $(_file)
exit
end

copy local=*.txt

forfiles remote=*.txt
   exec remote cmd=ksh input=mydata
end

close
```

15.11.2 Printing Data Element Information

Issuing the data command by itself prints a list of the names of all the data elements that have been defined.

Note: Data elements persist beyond individual UDM transfer sessions.

Issuing the data command with the **`print`** parameter and the name of a data element will print the data in that element.

Continuing with the previous example, issuing:

```
data print=mydata
```

Will produce the following output:

```
----> Begin 'mydata' <----
   grep "this is my sequence" $(_file)
   exit
---->  End 'mydata'  <----
```

CHAPTER 16

UDM Transfer Operations

16.1 Overview

This chapter provides information on Universal Data Mover (UDM) Transfer Operations.

See Chapter 17 Transfer Operations (z/OS-Specific) and Chapter 18 Transfer Operations (IBM i-Specific) for transfer information specific to those operating systems.

16.2 Transfer Sessions

16.2.1 Opening a Transfer Session

UDM transfer operations all occur within the context of a transfer session. This section details how to open a UDM session.

Opening a Two-Party Transfer Session

All sessions are established using the open command. At its simplest, the open command specifies the primary and secondary servers for the session:

```
open logical1=hostname logical2=hostname
```

In this example, **`logical1`** and **`logical2`** are the user-assigned logical names of the primary and secondary servers, respectively. Each of these parameters is set to the host name or IP address of the corresponding server.

For two-party transfer sessions, where the UDM Manager acts as the primary server, **`hostname`** is not the host address of the local machine (this would initiate a three-party transfer with the primary server running on the local machine). Instead, the host address is either the name **`local`** or the asterisk (**`*`**) character:

```
open machine1=* machine2=somentmachine
```

In this example, a two-party transfer session is established between the UDM Manager, acting as the primary server with the logical name **`machine1`**, and another machine with the host name **`somentmachine`**, with the logical name **`machine2`**.

An alternate method of establishing a two-party transfer is simply to give the secondary server as a parameter to the open command:

```
open machine2=somentmachine
```

In this example, a two-party transfer session is implied. In such cases, the logical name of the UDM Manager / primary server side of the transfer session always will be **`local`**.

Opening a Three-Party Transfer Session

A three-party transfer session can be opened using the same syntax as a two-party transfer session. However, both the primary and secondary servers must be specified explicitly, and the host name of the primary server must be a valid IP or host address:

```
open machine1=somemvsmachine machine2=somentmachine
```

In this example, a three-party transfer session is established between a machine with the host name **somemvsmachine**, given the logical name **machine1**, and a machine with the host name **somentmachine**, given the logical name **machine2**.

Stoneman's Tip

It is important to keep in mind
that the host name of the secondary transfer server
should be specified from the point of view of the primary server,
since it will be making the connection to the secondary server.

Depending on your network configuration,
the host name for the secondary server
might be different from the UDM Manager's perspective
than that of the primary server's.

16.2.2 Session Options

The examples given thus far show the simplest versions of the open command. Additional options can follow each server name, such as the port on which the Universal Broker is listening, the codepage that the server uses for text translation, authentication information, and references for a file from which these options are read (this file may be encrypted, if desired). At the end of the open command are optional parameters that specify the type of encryption and compression used for the data transfer operations.

(See Chapter 14 UDM Commands for detailed information on these parameters).

Stoneman's Tip

Unless otherwise specified, UDM transfers file data
using the SSL protocol and the **NULL-MD5** cipher suite.

If you do not want to take the performance hit of SSL,
and authentication of the transferred data is not required,
you may want **encrypt=NULL-NULL** specified as a session option.

However, the **NULL-NULL** cipher suite must be in the cipher list
for all UDM servers involved in the transfer.

16.2.3 Closing a Session

When all transfer operations have concluded, you can close a transfer session by issuing a close command. At this point, UDM is ready to initiate another transfer session.

Alternatively, if you want to exit UDM, you can issue a quit command, which closes the transfer session and exits the UDM Manager.

16.3 File Systems

16.3.1 File System Overview

Platforms can support one or more file systems or file access methods.

- UNIX and Windows support a single hierarchical file system.
- z/OS support three file systems (or file access methods) under UDM:
    1. DSN (data set name, the default when UDM is running under z/OS)
    2. DD (ddname defined by a JCL DD statement)
    3. HFS (the hierarchical file system supported by USS)

    (See Section 16.5 z/OS File System for detailed information on z/OS file systems.)
- IBM i supports two file systems under UDM:
    1. LIB (the default file system)
    2. HFS (limited to the root and QOpenSys file systems under IFS)

    (See Section 18.2 IBM i I/O for detailed information on IBM i file systems.)

All transfer operations on a given server will take place in the server's current file system. Both servers in a transfer session do not have to be in the same file system. UDM is capable of reformatting data between different file systems.

Stoneman's Tip

The default file system under z/OS is DSN,
even if the UDM Manager is executed
from USS (UNIX System Services).

16.3.2 Changing the Current File System

Changing the current file system on a server is a simple matter of executing the filesys command, which has the following format:

```
filesys logical_name[={dd|dsn|hfs|lib}]
```

In this format, the logical name refers to the logical name of the transfer server to send the filesys command. An optional file system (for example, z/OS's DD, DSN, or HFS) can be specified after the logical name to change the current file system on that server. Sending a filesys command with just a logical name returns the current file system of the server.

Note: A filesys value of **dd** is available only on z/OS manager for two-party transfer.

16.4 UDM Common File System

UDM provides a set of consistent capabilities for a diverse set of file systems on many different operating systems. UDM commands attempt to behave in a consistent and predictable manner regardless of the file system or operating system on which UDM is running. In order to do so, UDM behavior is based on a Common File System (CFS) model.

CFS is biased towards the hierarchical file systems found on UNIX, Windows, or HFS (z/OS or IBM i). CFS terminology and commands then are applied to each of the UDM-supported file systems on different operating systems.

16.4.1 Common File System Terminology

UDM attempts to make consist use of file system terminology so that it can be applied consistently to file systems that are not hierarchical.

Table 16.1, below, lists CFS terminology for hierarchical file systems like UNIX, Windows, and HFS:

CFS Term	Description
path	Name of a file, which may or may not include a directory. A path is either an absolute path or a relative path. Examples: • /home/homer/phone.txt • phone.txt • ../homer/phone.txt
absolute path	Full path name of a file, starting at the root directory, network point, or drive letter. Examples: • /home/homer/phone.txt • \\FILESERVER\homer\phone.txt • C:\program files\phone.txt
relative path	Path name of a file that is relative to the current working directory. Examples: • phone.txt • ./phone.txt • ../phone.txt • myfiles/phone.txt • ../homer/phone.txt

CFS Term	Description
file	Name of a file. All files are located in a directory. The name does not include a directory name. Examples: • phone.txt • editor.exe
directory	Name of a directory. The name does not include a file. It can be absolute or relative. Examples: • /home/homer • / • . • .. • C:\program files
current directory	Every program that runs on a hierarchical file system has a current directory, also known as the working directory. For most programs, this is the directory from which it was invoked.

Table 16.1 CFS Terminology for Hierarchical File Systems

16.5 z/OS File System

The z/OS data set file system is a flat file system. There are no concepts of directories. The files are more commonly referred to as data sets.

z/OS data sets supported by UDM fall into two major categories:

- Sequential
  Sequential data set has a data set organization of Physically Sequential (PS).
- Partitioned
  Partitioned Data Set (PDS) has a data organization of Partitioned Organization (PO), which also includes system managed Partitioned Data Set Extended (PDSE) organization.

A PDS is treated as a directory in CFS. A PDS contains a set of individual files called members, which is analogous to a directory containing a set of files. A PDS member has a maximum length of 8 characters.

Table 16.2, below, associates CFS terminology with z/OS partitioned and sequential data sets. Fully qualified data set names are enclosed in apostrophes.

CFS	Sequential	Partitioned
path	Data set name. A path includes fully qualified names and relative names. Examples: • PHONE.DATA • 'MYUID.PHONE.DATA'	PDS name and member name in parenthesis or just a member name. A path includes fully qualified names and relative names. Examples: • JCL.CNTL(JOBAB) • 'MYUID.JCL.CNTL(JOBAB)' • JOBAB
absolute path	Fully qualified data set name. Example: • 'MYUID.PHONE.DATA'	Fully qualified PDS name and member name in parenthesis. Example: • 'MYUID.JCL.CNTL(JOBAB)'
relative path	Data set name without one or more leading qualifiers. The name is relative to the current directory. Examples: • PHONE.DATA • DATA	PDS name without its high-level or mid-level qualifiers and a member name enclosed in parenthesis or just a member name. The name is relative to the current directory. Examples: • JCL.CNTL(JOBAB) • CNTL(JOBAB) • JOBAB
file	Same as path (data set name). It may be absolute or relative.	Member name only. Example: • JOBAB

CFS	Sequential	Partitioned
directory	N/A	PDS name without the member name. The directory name may be relative or absolute. Examples: • JCL.CNTL • 'MYUID.JCL.CNTL'
current directory	Current leading qualifiers. **Note:** It may be more than one qualifier long. Examples: • MYUID • MYUID.DATA	Current leading qualifiers. **Note:** It may be more than one qualifier long or even the full PDS name. Examples: • MYUID • MYUID.JCL • 'MYUID.JCL.CNTL'

Table 16.2 CFS Terminology Associated with z/OS Data Sets

UDM is capable of running as a JES batch job. In a batch environment data sets may be allocated dynamically by UDM or UDM may use data sets pre-allocated with JCL DD statements.

The JCL DD statement allocates the data set and defines its to the batch job environment as a ddname that the program uses. Although ddnames are not a different file system, they do have their own naming conventions and behavior relative to UDM's CFS.

Table 16.3, below, associates CFS terminology with z/OS partitioned and sequential data sets allocated to ddnames.

CFS	Sequential	Partitioned
path	ddname defined with a JCL DD statement. Examples: • DD1 • PHONE	ddname with a member name enclosed in parenthesis or just the member name. Examples: • INDD(JOBAB) • MYDATA(PHONE) • PHONE
absolute path	There is only one type of path and that is absolute. Refer to path above. • DD1	ddname and member name enclosed in parenthesis. Refer to path above.
relative path	N/A	Member name only. The path is relative to the current directory.
file	Same as path (the ddname). Example: • DD1	Member name only. Example: • JOBAB
directory	N/A	ddname with which a PDS is allocated.
current directory	N/A	Current ddname with which a PDS is allocated. Examples: • `INDD` • `OUTDD`

Table 16.3 CFS Terminology Associated with z/OS ddnames

16.6 IBM i File Systems

Universal Data Mover for IBM i supports two types of file systems:

- LIB (library) file system supports the original, native database file system.
- HFS file system supports the root and QOpenSys file systems under IFS.

Although UDM can access other IFS file systems, only root and QOpenSys are certified.

Currently, Stonebranch, Inc.:

- Does not support other IFS file systems.
- Recommends that users do not use other IFS file systems.
- Provides no warranty for use of other IFS file systems.
- Certifies that users assume all risks in using other IFS file systems.

Risks involved in the use of non-supported IFS files systems include, but are not limited to:

- Loss of data
- Corrupted data
- Non-recoverable exceptions

16.6.1 HFS

HFS follows the common file system (CFS). It supports stream files under the root and QOpenSys IFS file systems. Users using UDM to access file systems under IFS, other than root and QOpenSys, do so at their own risk.

16.6.2 LIB

UDM for IBM i supports the following file types of the LIB (library) file system

- Physical files (source and data)
- Save files

Table 16.4, below, associates CFS terminology with these LIB file types.

CFS	Physical Files (Source and Data)	Save Files
path	An absolute or relative path. Examples: • MYLIB/DATA(NAMES) • DATA(NAMES)	An absolute or relative path. Examples: • MYLIB/BACKUP • BACKUP
absolute path	A fully qualified name containing a library, file, and member. Example: • MYLIB/DATA(NAMES)	A fully qualified name containing a library and file. Example: • MYLIB/BACKUP
relative path	A name without a library. The name is relative to the current directory (library). Example: • DATA(NAMES)	A name without a library. The name is relative to the current directory (library). Example: • BACKUP
file	Same a path. It may be relative or absolute.	Same as path. It may be relative or absolute.
directory	N/A	N/A
current directory	Name of the current library in which you are working. Example: • MYLIB	Name of the current library in which you are working. Example: • MYLIB

Table 16.4 CFS Terminology Associated with LIB File Types

16.7 Transfer Modes and Attributes

16.7.1 Setting the Transfer Type

There are two basic types of file transfers:

- Binary
  Binary transfers move the data as it is, without any translation.
- Text
  Text transfers translate the data from the source server's code page to the destination server's code page as it is transferred from one server to another.

The default transfer type for UDM is binary.

To set the transfer type, use the mode command.

- To set up UDM for text transfers, issue the following command:

  ```
  mode type=text
  ```

- To set up UDM for binary transfers, issue the following command:

  ```
  mode type=binary
  ```

Issuing the mode command by itself displays the current transfer mode. The mode command also can be used tell UDM to trim trailing spaces at the end of each line (or record, for record-based file systems such as **dd** and **dsn** in z/OS).

16.7.2 Transfer Attributes

While the mode command is used to control the settings for transfer operations as a whole, the attrib command can be used to set up the handling of transfer operations for each side of the transfer session.

The attrib command can set transfer attributes that apply to either the primary or secondary server. It takes the following form:

```
attrib lname[={dd|dsn|hfs}] [attribute 1=value1]…[attribute n=valuen]
```

Where **lname** is the logical name of the server, the attributes are to be applied.

By default, any attributes listed in the attrib command are applied to the currently selected files system unless a specific file system is assigned to the logical name. In that case, the attributes are applied to the specified file system.

The remainder of the attrib command contains a series of attributes and their values, some of which will be discussed in further detail in the remainder of this section. If the attrib command is issued with just a logical name, UDM will list the currently set attributes for the corresponding server.

When you change file systems for a server using the **filesys** command, the currently set attributes are those that were applied to that file system type.

In other words, attributes are not carried over from one file system to another.

Stoneman's Tip

16.7.3 End of Line Sequence

Text mode transfers have the concept of a line in UDM. For record-oriented file systems, such as z/OS's DD and DSN, and IBM i's LIB, each line is a single record. However, for UNIX, Windows, and the HFS file system under USS and IBM i, there is no inherent structure imposed by the operating system on file data.

To determine what constitutes a line in the data for these types of files, UDM looks for an end of line sequence on the source side of a transfer. This can be any sequence of characters (including a zero length sequence, in which case the entire file is considered to be a single line). UDM determines when it has read a complete line of data when this sequence is encountered.

In addition to the normal printable character sets on each platform, an end of line sequence also can be:

- `\r` character sequence, to denote a carriage return character.
- `\l` sequence, to denote a line feed.
- `\n` sequence, to indicate a new line character.

Stoneman's Tip

When UDM transfers a line of text data from one server to another, it does not transfer the end of line sequence.

Instead, UDM transfers all of the data in each line up to the end of line sequence.

The end of line sequence also is used on the destination side of a text transfer. The end of line sequence set for the destination side of the transfer is appended to the end of each line of data.

UDM also does this for record-oriented file systems as well. By managing the end of line sequence this way, UDM easily can be used to translate end of line characters across platforms (such as a transfer from UNIX to Windows), strip end of line characters from the data completely, or even add a completely new end of line sequence for use by other applications. For most operations, though, the end of line sequence will not need to be changed.

eol Attribute

The end of line sequence is set with the **eol** attribute. The default value for **eol** depends on the platform and file system selected:

- For Windows-based platforms, the default value is **\r\n**.
- For UNIX platforms and the HFS file system under USS, the default value is **\n**.
- For the HFS file system under IBM i, the default is FILE, which makes end of line terminator consistent with file CCSID.
- For record-oriented file systems (z/OS's dd and dsn, and IBM i's LIB), the value for **eol** is not set.

To provide consistent **eol** definitions under the IBM i HFS file system, specific ASCII and EBCDIC values are defined for the symbolic values.

- As ASCII, \n = x0A, \r = x0D, \t = x09 and \l = x0A.
- As EBCDIC, \n = x15, \r = x0D, \t = x05 and \l = x25.

By default, the file CCSID determines the type of **eol**, ASCII vs. EBCDIC. The default ASCII **eol** is **\n** and the default EBCDIC **eol** is **\r\l**.

It is important to note the difference between **eol** definitions as just described and **eol** characters when transferred as data. Due to code page translations and Unicode mappings that take place during data transfer, translated values may be surprising.

Please refer to appropriate translation tables or Unicode mapping tables to understand the values used when **eol** and other control characters are transferred as data. UDM provides default definitions and allows user-defined **eol** attribute overrides in order to avoid translation surprises and associated difficulties.

The following example sets an end of line sequence of an exclamation point (**!**) for a transfer server:

```
 attrib mylogicalname eol=!
```

16.7.4 Line Length and Line Operations

Note: The attributes discussed in this subsection apply solely to the destination side of the transfer.

Other attributes can be used to manipulate transferred data as well.

The **`linelen`** attribute is used to specify the length, in characters, of a line of data that has been transferred. This value is independent of the end of line sequence and, for record-oriented file systems, the transfer type. If **`linelen`** is set to a value other than *zero* (its default value), UDM will manipulate the data according to the method specified with the **`lineop`** attribute.

The **`lineop`** attribute specifies what happens to each line (or record, from z/OS's dd and dsn file systems) of data coming from the source transfer. If the value for **`lineop`** is *none*, the line/record is written as is, however if its length from the source is greater than the value of **`linelen`**, UDM issues an error. If the value of **`lineop`** is *stream*, the data from the source side of the transfer is treated as a single record and is subdivided when it is written as a series of lines or records (depending on the file system), each **`linelen`** characters in length. If the value of **`lineop`** is *trunc*, each record or line from the source is truncated so that it is at most **`linelen`** characters in length. Finally, if the value of **`lineop`** is *wrap*, each line or record from the source side of the transfer that is longer than **`linelen`** characters is wrapped into multiple lines/records so that the maximum length of each line on the destination side is at most **`linelen`** characters long.

Stoneman's Tip

Binary data that is transferred from a Windows or UNIX platform
(including HFS under USS)
is looked at by UDM as one large line or record of source data.

The same can be said when transferring text data from these platforms
if the end of line sequence is zero length for the source server
or the end of line sequence does not exist in the source data.

Under z/OS (except for the HFS file system), UDM will set the **`linelen`** attribute to be the same as the **`lrecl`** allocation option for new data sets or the LRECL DCB attribute of existing data sets if the value of **`linelen`** is *zero*. UDM also will set the **`lineop`** attribute to a value appropriate for the transfer type and destination allocation attributes if **`lineop`** has previous not be set.

16.8 Copying Files with UDM

16.8.1 Simple Copy Operation

At its core, UDM is meant to copy files from one system to another. This is done with the copy command.

The basic format of the copy command is:

```
copy sourcelname=filespec [destlname[=filespec]]
```

The copy command copies the file specified on the server with the logical name corresponding to **sourcelname** to the server with the logical name corresponding to **destlname**.

If no destination file name is given, the source file name is used (absent the directory name, regardless of whether or not it was explicitly specified as part of the source file specification). Likewise, if only a directory is given for the destination file specification, the source file name is appended to the directory when writing the file.

If a destination file name or complete file name and directory are given for the destination file specification, that information is used in writing the destination file, regardless of the source file name.

Examples

The following example copies the file **test.txt** from a machine with the logical name **src** to a file called **test.txt** on the machine with the logical name **dst**:

```
copy src=test.txt dst
```

The following example copies the test.txt file residing in **c:\files** from a machine with the logical name **src** to a file called **test.txt** on the machine with the logical name **dst**:

```
copy src=c:\files\test.txt dst
```

The following example copies **test.txt** from **src** to the root of drive C on **dst**. The destination file is also called **test.txt**:

```
copy src=test.txt dst=c:\
```

The following example copies the file **test.txt** from **src** to a file called **test.bak** in the root of drive C on **dst**:

```
copy src=test.txt dst=c:\test.bak
```

Stoneman's Tip

If you want the destination file name
to be the same as that of the source file name,
you do not have to specify the destination system
in the **copy** command.

The destination will be implied based on the logical name of the source
(if the source is the primary server,
the destination is assumed to be the secondary server,
and vice versa).

16.8.2 Move Operation

A UDM move operation, using the move command, is similar to a copy operation.

The only difference between using a move command and a copy command is that after you move a file, it is deleted from the source server from which it was moved.

16.8.3 Copying Multiple Files Using Wildcards

In addition to copying single files, the copy command can be used to copy multiple files by using wildcards in the source file specification. When wildcards are used, any file matching the source file specification will be copied.

There are two types of wildcards:

1. Asterisk (*) wildcard matches zero or more characters.
2. Question mark (?) wildcard matches a single character.

Here are some examples of wildcard matching given the following filenames in the source directory:

- **`test.txt`**
- **`test1.txt`**
- **`test2.txt`**
- **`test3.txt`**
- **`test.bin`**
- **`test1.bin`**
- **`test2.bin`**
- **`test3.bin`**

A source file specification of `*` will copy all of the files in the directory, as will `*.*`, `test*.*`, and `tes?.*`.

A source file specification of `*.txt` will copy the first three files.

A source file specification of `*.bin` will copy the last three files.

A source file specification of `test?.txt` will copy the files `test1.txt`, `test2.txt`, and `test3.txt`.

Stoneman's Tip

Wildcards can be used only in the source file specification, not the destination file specification.

Under some operating systems, it is possible for * and ? to be valid characters in a filename. When they appear in the destination portion of a UDM copy operation, they are treated as file characters and not as wild cards.

Also, keep in mind that while UDM can copy all of the files at a single directory level in a hierarchical file system, it will not traverse the directory tree and copy files from directories at a lower level than the current directory or the directory explicitly specified in the source file specification.

Wildcards should appear only in the filename portion of the file specification and not as part of the directory itself.

16.8.4 File Extension Attributes

Stoneman's Tip

File extension attributes only come into play
when a destination filename is not specified.

UDM considers a file extension to be the character sequence
followed by the last period (.) in the filename,
not including a dot character appearing as the first character in the filename.

The character sequence specified by **`defaultext`** is appended verbatim.

A dot (.) is not implied at the beginning of this sequence
and must be explicitly included
if it is desired in the destination filename.

Some file systems support file extensions. In the case where the source filename is used as the destination filename, UDM can either add an extension or truncate a file's extension. If the **`trunctext`** attribute is set to *yes*, the extension of the source filename is truncated when writing the destination file. If the **`defaultext`** attribute is set to any value, that value is appended to the end of the source filename when writing the destination file.

16.8.5 File Creation Options

The **`createop`** attribute determines how the destination file is created. By default it has a value of *new*, which means that a file with the destination filename (either implicitly or explicitly specified) must not exist for it to be successfully written by UDM. If a file with that name does exist when UDM begins a copy operation, an error is issued.

If the value of the **`createop`** attribute is replace, the destination file is created if the destination file does not already exist. If it does exist, it is overwritten with the transferred data. If the value of **`createop`** is append and the file already exists, the data transferred is appended to the end of the data already in the existing file. If the file does not exist, a new file is created.

16.8.6 File Permission Attribute

Under the UNIX operating system, the **mode** attribute specifies the mode (in UNIX parlance), or file permissions, of a file created by UDM in a copy operation. Existing files do not have their modes modified by UDM. They retain the file mode that they had before the copy operation was initiated.

Note: The **mode** attribute is not to be confused with the mode command, which is used to set the type of file transferred (and trim option).

The **mode** attribute is set using the attrib command.

The value of **mode** is either a set of three numbers, or nothing. Each number in the set corresponds to one or more individuals for whom access is granted for the file:

• First number	Owner of the file.
• Second number	Users in the group assigned to the file.
• Third number	Everyone else.

The value of each number is the sum of values representing file permissions:

• 0	No permissions.
• 1	Permission to execute the file.
• 2	Permission to write to the file.
• 4	Permission to read from the file.

Thus, a value of 7 for the first number would provide the file owner with permission to read, write, and execute a file. A value of 6 for the second number would provide users in the group assigned to the file with permission to read from the file and write to the file, but not to execute the file. A value of 0 for the third number would provide everyone else with no permissions for the file.

By default, the mode attribute is not set. The default mode of a newly created file by UDM is dependent upon the user's umask or the mode of the source file in a UDM transfer.

Examples

The following example provides the owner, group, and everyone else permission to read, write, and execute the file:

```
attrib local mode=777 (
```

The following example provides the owner permission to read, write, and execute the file; members of the file's group permission to read and execute the file; and everyone else no permissions.

```
attrib local mode=750
```

The following example provides the owner permission to read and write the file; and no permissions to the file's group and everyone else.

```
attrib local mode=600
```

Defaults

By default, the **mode** attribute is not set.

The default mode of a newly created file by UDM is dependent upon either:

- User's **umask** attribute.
- Mode of the source file in a UDM transfer.

The latter case comes into play if both the source and destination instances of UDM are:

- Version 3.2 or greater.
- Running under some form of the UNIX operating system or its derivatives (such as Linux).

The **umask** attribute is used in specifying the mode if a UDM version prior to 3.2.0 is involved in the transfer. Version 3.2.0 (and greater) versions can be changed to behave this way as well by setting the **mode** attribute to 0 on the destination side of the transfer.

For example:

```
attrib dest_logical_name mode=0
```

16.8.7 Destination umask

Under UNIX platforms and the HFS file system under z/OS USS, the **umask** attribute can be used to define the file's permissions in accordance to UNIX standards (see Defaults in Section 16.8.6 File Permission Attribute, above).

See Table 21.2 Common File System Attributes for a detailed description of **umask**.

16.8.8 Transaction-Oriented Transfers

A transaction oriented transfer is a file transfer where the destination file is written using a temporary filename. Once the file transfer has been completed, the file is renamed to the appropriate destination filename. To turn on transaction-oriented transfers, set the **trans** attribute for the server on the destination side of the transfer to yes.

Note: UDM for IBM i LIB file system does not support transaction-oriented transfers.

16.8.9 Changing the Current Directory in UDM

The **cd** command is an easy way to change the current directory in the UDM Common File System, discussed earlier in this section.

By default, when the manager is involved in a two-party transfer, the current directory for primary server is the path in which the manager was launched under. Under z/OS, this would be the high-level qualifier for the user id the manager is running as. The secondary server (as well as the primary server in a three-party transfer) has a default path of the authenticated user's home directory for hierarchical file systems and a high level qualifier corresponding to the authenticated user for dd and dsn file systems.

A user can change the current path for a specific server by issuing a **cd** command:

```
cd lname[=current-directory]
```

In this example:

- **lname** is the logical name of the transfer server to change its default path.
- **current-directory** is the new current directory to set.

If the **cd** command is issued with only a logical name, UDM displays the current directory for the corresponding transfer server.

16.9 Auditing Transfer Operations

16.9.1 Logging File Transfer Operations

When the message level in the UDM server's configuration is set to **audit**, the server writes audit messages to the broker log for each file transferred.

The following is an example of the audit messages produced:

```
NV3950A [1110470739] Transferring from: host: 'Enderlyn.local'
(10.0.0.101), user: 'root', file: '/Volumes/Archive/VPC
Images/.DS_Store'

UNV3951A [1110470739] Transferring to:   host: 'Aluminum.local'
(10.0.0.100), user: 'kevin', file: '/Users/kevin/Desktop/tmp/.DS_Store'

UNV3952A [1110470739] Successfully transferred '/Volumes/Archive/VPC
Images/.DS_Store' on 'Enderlyn.local' to
'/Users/kevin/Desktop/tmp/.DS_Store' on 'Aluminum.local'
```

The first message is written when the transfer server receives a request to initiate a file transfer and contains the host name of the source machine, the IP address of the source machine, the user authenticated with UDM at the source of the transfer, and the name of the source file to be transferred.

The second message is written when the destination transfer server acknowledges the file transfer requested and contains the host name of the destination machine, the IP address of the destination machine, the user authenticated with UDM at the destination of the transfer, and the destination filename that will be used.

The third message produced indicates that the file was transferred successfully. This message contains the source and destination filenames and host names.

The UDM Manager also can produce these messages when it is involved in a two-party transfer session (though much of the information will be redundant with its standard information messages) by setting its message level to audit. The manager's audit messages are written to stderr (sysout under z/OS).

16.9.2 Reporting Transfer Progress

For long transfer operations, it is often useful to see periodic indications of the operation's progress.

You can get this information by turning on progress reporting using the report command:

```
report progress=yes
```

This will cause the UDM manager to issue periodic updates regarding the progress of a file being transferred. The interval these updates are given is the same as the keep alive interval.

Progress messages look as follows:

```
1024000 bytes processed
```

CHAPTER 17

Transfer Operations (z/OS-Specific)

17.1 Overview

This chapter provides information on Universal Data Mover (UDM) transfer operations that are specific to the z/OS operating system.

- z/OS I/O
- UDM Commands under z/OS
- Copying Load Modules

17.2 z/OS I/O

This section provides an overview of the z/OS file systems.

17.2.1 Data Sets

There are a variety of data sets on z/OS. The UDM-supported data set organizations and data set attributes are listed below.

Data Set Names

A z/OS data set name is composed of one or more qualifiers separated by periods. A data set has a maximum length of 44 characters.

A qualifier has an maximum length of eight characters. The first character of a qualifier must start with A-Z, @, #, or $. The remaining characters can be 0-9, A-Z, @, #, $, or -.

The first qualifier is commonly referred to as the high-level qualifier (HLQ).

An example data set name is **`SYS1.CEE.CEEPRC`**, where:

- **`SYS1`** is the high-level qualifier.
- **`CEE`** is the second qualifier.
- **`CEEPRC`** is the third and last qualifier.

In some applications contexts, the HLQ can be left off. TSO and ISPF are such applications. UDM also behaves in this manner. A distinction is the made between a data set that specifies the HLQ and one that does not specify the HLQ. When the HLQ is specified, it is referred to as a fully qualified data set name and is enclosed in apostrophes.

Data Set Organization

A data set's organization is obtained from the VTOC's format-1 DSCB. If the data set is cataloged, the DSCB is only read for non-VSAM catalog entries.

The following organizations are supported:

- Physical Sequential
- Partitioned Organization
- Partitioned Data Set Extended
- Generation Data Set

The following organizations are not supported:

- Indexed sequential
- Direct
- Unmovable
- VSAM

Any organization not listed is undetermined.

Record Format

The following record formats are supported:

- Fixed length
- Variable length
- Undefined length
- Blocked
- Fixed length standard
- Variable length spanned
- ISO/ANSI control character
- Machine control character

Block Size

There are three different types of block sizes:

1. User specified block size that cannot exceed 32,760.
2. System determined block size that cannot exceed 32,760.
3. Block sizes supported by the Large Block Interface (LBI) that permits sizes up to 2G. LBI is supported by DFP on tape devices only at this time.

17.2.2 Generation Data Group and Generation Data Sets

A Generation Data Group (GDG) is a catalog entry used to maintain a group of Generation Data Sets (GDS).

GDS's are referred to with absolute names or relative names:

- Absolute name has the form of GDG.G0000V00.
- Relative name has the form of GDG(n), where:
  - n = 0 for the current GDS
  - n = -1 for the previous GDS
  - n = +1 for a new GDS

Allocation

Allocation attributes for a GDS are obtained differently depending on whether the data set name is an absolute or relative form.

Absolute Name

Allocation attributes for an absolute name are provided like any other data set through JCL keywords or allocation options.

Relative Name

Allocation attributes for relative names are provided with one of the following methods:

1. By referring to a cataloged data set from which attributes are copied.
   a. DCB=(dsname)
   b. LIKE=dsname
   c. REFDD=ddname
2. By referring to a model Data Set Control Block (DSCB) on the volume on which the GDG is cataloged. This cannot be used for SMS managed data sets.
   a. DCB=(modeldsname,yourattributes)
3. By using the DATACLAS and LIKE allocation keywords.
4. Through the assignment of a DATACLAS by a data class ACS routine.

17.2.3 Catalogs

There are two types of catalogs:

1. Integrated Catalog Facility (ICF)
2. VSAM Catalogs

Note: IBM has dropped support for VSAM Catalogs as of January 1, 2000; UDM does not support them.

Symbolic Names

A catalog entry can be defined with symbolic names for the volume serial number. UDM does resolve the symbolic names when they are found for the volume serial number.

Catalog Entry Types

A catalog entry is defined as a specific type. UDM only supports the non-VSAM type entry. A catalog entry type can by any one of the following:

- Non-VSAM Data Set
- Generation Data Set
- Cluster
- Alternate Index
- VSAM Path
- Alias
- User Catalog Connector
- Tape Volume Catalog Library
- Tape Volume Catalog Volume

17.2.4 Allocation

Data set allocation is the process of obtaining access to the data set.

If the data set already exists, it resides on a device, such as a tape or, more likely, a disk. In order to allocate an existing data set, the device must be known. A volume serial name or number and a unit name or number represents an I/O device in z/OS.

The unit and volume serial number (`volser`) can be specified explicitly or specified implicitly (with a catalog entry).

Allocation can be performed with JCL or dynamically. Dynamic allocation requires allocation attributes to be specified by the user. UDM dynamic allocation of a data set that has been migrated by HSM (or similar three-party product) will result in the data set being recalled. UDM will wait until the recall is complete and then continue processing.

17.3 UDM Commands under z/OS

This section describes the behavior of UDM commands when working with z/OS data sets and ddnames.

17.3.1 attrib (Attribute) Command

z/OS data sets can be allocated statically with JCL DD statements or dynamically with the z/OS Dynamic Allocation service (aka SVC 99).

Table 17.1, below, lists the dynamic allocation attributes that can be specified via the attrib (Attribute) command. For complete details on an allocation attribute, refer to the IBM JCL Reference.

Note: When performing a z/OS-to-z/OS copy and the destination file system is DSN, UDM uses the following allocation attributes obtained from the source file (assuming the file system is DD or DSN): blksize, dirblocks, dsorg, lrecl, primspace, secspace, and spaceunit.

If you do not want to use the source files values for these allocation attributes, you can override them by issuing an attrib command with the attributes that you want to change before the copy operation.

Attribute Name	Description
abnormaldisp	• Disposition of a data set after the job ends abnormally. • Equivalent to the third position sub-parameter of the JCL DISP parameter: DISP=(status,normaldisp,abnormaldisp). • Default is DELETE: it can be set with the UDM configuration option ALLOC_ABNORMAL_DISP.
avgrec	• Indication that the unit of allocation space specified with the **`spaceunit`** attribute is records and that the primary space and secondary space values are in units of 1's, K's, or M's. • Equivalent to the JCL AVGREC parameter: AVGREC=size. • No default or configuration option.
blksize	• Block size with which the data set is allocated. • Equivalent to the JCL BLKSIZE parameter: BLKSIZE=size. • Default is 27998: it can be set with the UDM configuration option ALLOC_BLKSIZE.
blkszlim	• Block size limit when there is not block size specified from any source. • Equivalent to the JCL BLKSZLIM parameter: BLKSZLIM=size. • No default or UDM configuration option.
dataclas	• SMS data class name. • Equivalent to the JCL DATACLAS parameter: DATACLAS=name. • No default: it can be set with the UDM configuration option ALLOC_DATACLAS.

Attribute Name	Description
datasetseq	• Data set sequence number that specifies the relative position of a tape data set on the volume. • Equivalent to the data set sequence sub-parameter of the JCL LABEL parameter: LABEL=(datasetseq,,,,). • No default or UDM configuration option.
ddndcbref	• DCB reference to a ddname. • Equivalent to the ddname sub-parameter of the JCL DCB parameter: DCB=ddname. • No default or UDM configuration option.
den	• Tape density to use. • Equivalent to the DEN sub-parameter of the JCL DCB parameter: DCB=DEN=density. • No default or UDM configuration option.
dirblocks	• Number of directory blocks to allocate for a partitioned data set. • Equivalent to the third positional parameter of the second positional parameter of the JCL SPACE parameter: SPACE=(, (,, dirblocks), . . .). • Default is 20: it can be set with the UDM configuration option ALLOC_DIR_BLOCKS.
dsndcbref	• DCB reference to a cataloged data set name. • Equivalent to the data set name sub-parameter of the JCL DCB parameter: DCB=dsn. • No default or UDM configuration option.
dsntype	• Type of SMS data set to allocate. • Equivalent to the JCL DSNTYPE parameter: DSNTYPE=type. • No default or UDM configuration option.
dsorg	• Data set organization with which the data set is allocated. • Equivalent to the JCL DSORG parameter: DSORG=org. • Default is PS: it can be set with the UDM configuration option ALLOC_DSORG.
expdt	• Expiration date of the data set. • Equivalent to the JCL EXPDT parameter: EXPDT=date. • No default or UDM configuration option.
label	• Data set label type used for mostly tape data sets. • Equivalent to the label sub-parameter of the JCL LABEL parameter: LABEL=(,label,,,). • No default or configuration option.
like	• SMS data set name from which to model data set attributes. • Equivalent to the JCL LIKE parameter: LIKE=dsname. • No default or UDM configuration option.
lrecl	• Logical record length with which the data set is allocated. • Equivalent to the JCL LRECL parameter: LRECL=len. • Default is 1024: it can be set with the UDM configuration option ALLOC_LRECL.
mgmtclas	• SMS management class name. • Equivalent to the JCL MGMTCLAS parameter: MGMTCLAS=name. • No default: it can be set with the UDM configuration option ALLOC_MGMTCLAS.
normaldisp	• Disposition of a data set after the job ends. • Equivalent to the JCL DISP parameter: DISP=(status,normaldisp,abnormaldisp). • Default is CATLG: it can be set with the UDM configuration option ALLOC_NORMAL_DISP.

Attribute Name	Description
password	• Password for password protected data sets. • No JCL equivalent.
primspace	• Primary amount of space to allocate for the data set. • Equivalent to the first positional parameter of the second positional parameter of the JCL SPACE parameter: SPACE=(, (primspace,), . . .). • Default is 15: it can be set with the UDM configuration option ALLOC_PRIM_SPACE.
pwdprotect	• Specification for whether or not the data set is password protected. • Equivalent to the PASSWORD or NOPWREAD sub-parameters of the JCL LABEL parameter: LABEL=(,,{PASSWORD|NOPWREAD},,). • Value must be either PASSWORD or NOPWREAD.
recfm	• Record format with which the data set is allocated. • Equivalent to the JCL RECFM parameter: RECFM=fmt. • Default is VB: it can be set with the UDM configuration option ALLOC_RECFM.
refdd	• ddname from which to copy SMS data set attributes. • Equivalent to the JCL REFDD parameter: REFDD=ddname. • No default or configuration option.
retpd	• Retention period of the data set. • Equivalent to the JCL RETPD parameter: RETPD=date. • No default or configuration option.
rlse	• Specification for whether or not to release unused space when the data set is unallocated. • Equivalent to the sub-parameter RLSE of the JCL SPACE parameter: SPACE=(,(,,),RLSE). • Default is no. There is no UDM configuration option. • Setting the attribute value to *yes* turns on the attribute.
secspace	• Secondary amount of space to allocate for the data set. • Equivalent to the second positional parameter of the second positional parameter of the JCL SPACE parameter: SPACE=(, (, secspace), . . .). • Default is 15: it can be set with the UDM configuration option ALLOC_SEC_SPACE.
spaceunit	• Allocation unit used to specify the space to allocate for the data set. • Equivalent to the first positional parameter of the JCL SPACE parameter: SPACE=(unit, . . .). • Default is TRK: it can be set with the UDM configuration option ALLOC_SPACE_UNIT.
status	• Status of the data set to be allocated. • Equivalent to the first positional parameter of the JCL DISP parameter: DISP=(status,normaldisp,abnormaldisp). • Default is: OLD for input and output data sets that exist, NEW for output data sets that don't exist. • Default input status can be set with UDM configuration option ALLOC_INPUT_STATUS. • Default output status can be set with UDM configuration option ALLOC_OUTPUT_STATUS.
storclas	• SMS storage class name. • Equivalent to the JCL STORCLAS parameter: STORCLAS=name. • No default: default value can be set with UDM configuration option ALLOC_STORCLAS.

Attribute Name	Description
unit	• Unit on which the data set is allocated. • Equivalent to the JCL UNIT parameter: UNIT=unit. • Default is SYSALLDA: it can be set with UDM configuration option ALLOC_UNIT.
unitcnt	• Number of units to allocate for a multi-volume data set. • Equivalent to the unit count sub-parameter JCL UNIT parameter: UNIT=(,unitcnt,). • No default or UDM configuration option.
volcnt	• Number of volumes to allocate for a multi-volume data set. • Equivalent to the volume count sub-parameter JCL VOL parameter: VOL=(,,,volcnt,). • No default or UDM configuration option.
volseq	• Volume sequence number on which a multi-volume data set starts. • Equivalent to the volume sequence number sub-parameter JCL VOL parameter: VOL=(,,volseq,,). • No default or UDM configuration option.
volser	• Volume serial number on which the data set is allocated. • Equivalent to the SER sub-parameter of the JCL VOL parameter: VOL=SER=volser. • No default: default value can be set with UDM configuration option ALLOC_VOLSER.

Table 17.1 attrib Command - Dynamic Allocation Attributes

17.3.2 cd (Change Directory) Command

The cd (Change Directory) command moves the current position within a file system. Position means different things depending on the file system.

This section describes the behavior of the cd command for each file system.

DSN (data set name) File System

The DSN (data set name) file system has no directories. The cd command treats each data set qualifier as a directory in regards to traversing and positioning within the data set name space.

UDM initializes the current directory to a high-level qualifier equal to the user identifier with which UDM executes.

A cd value can be enclosed in apostrophes (**'**). One or more qualifiers enclosed in apostrophes replaces the current directory value.

A cd value not enclosed in apostrophes is concatenated to the current directory value separated by a period (**.**), effectively moving up in the hierarchy.

There are two special directory (qualifier) names:

1. Current directory - represented by a single period (**.**)
   Directory name **.** makes no change.
2. Previous directory - represented by two periods (**..**).
   Directory name **..** moves back one qualifier.

Examples

Table 17.2, below, provides examples of positioning within the data set file system using the cd command. The examples assume the following:

- User ID of TOM123
- UDM logical name SRV

Current Directory before cd	cd command	Current Directory after cd
`TOM123`	`cd srv=data`	`TOM123.DATA`
`TOM123.DATA`	`cd srv=appl.jcl`	`TOM123.DATA.APP1.JCL`
`TOM123.DATA.APP1.JCL`	`cd srv=..`	`TOM123.DATA.APP1`
`TOM123.DATA.APP1`	`cd srv='GAM789.DATA'`	`GAM789.DATA`
`GAM789.DATA`	`cd srv=..`	`GAM789`
`GAM789`	`cd srv=..`	`GAM789`

Table 17.2 cd Command in DSN File System

DD (ddname) File System

The DD (ddname) file system, like the DSN file system, has no directories. DD is the simplest form of file system in UDM.

A ddname is defined with a JCL DD statement. All the allocation attributes are specified with on the JCL DD statement.

UDM initializes the current directory to blanks in the DD file system.

A cd value specifies an allocated ddname to use as the current directory.

17.3.3 copy (Copy) Command

The copy command copies files between two systems. The source and destination files are specified with a file specification. The file specification syntax depends on the file system being referenced. This section describes the syntax and semantics of the copy command's file specification.

z/OS

The copy command also can be used to copy load modules (see Section 17.4 Copying Load Modules).

DSN File System

The semantics of a file specification is determined primarily by whether a sequential or a partitioned file is being referenced. A sequential file is treated as a single entity in regards to reading and writing. A partitioned file is treated as a composite of multiple sequential files each operated on individually.

Sequential Data Sets

A file is considered sequential if it has a data set organization of Physical Sequential (PS).

A file is referenced directly as a fully qualified name enclosed in apostrophes (') or as a relative name composed of one or more qualifiers concatenated to the current working directory value to form a fully qualified name. The qualifiers **.** and **..** , which are used in the cd command, do not have any special meaning in a file specification. They most likely will result in an invalid fully-qualified data set name.

Included in the sequential category are generation data sets. A data set is considered a generation data set if it had a generation data group catalog entry and the data set name includes a generation relative number (for example: (0), (+1), (-1)).

Table 17.3, below, provides some examples of **copy** command file specifications for sequential data sets. The examples assume a UDM logical name of SRV.

Current Directory	File Specification	Result
`TOM123`	`copy srv=data`	`TOM123.DATA`
`TOM123`	`copy srv=app1.jcl`	`TOM123.APP1.JCL`
`TOM123`	`copy srv='GAM789.APP2.DATA'`	`GAM789.APP2.DATA`

Table 17.3 copy Command File Specifications for Sequential Data Sets

Note: In the case of a destination file specification, if no destination file is specified and the attribute **usefqn** is set to *no* (default) for the source **dsn** transfer server, only the part of the data set name matching the source mask in the copy operation is used as the destination file name. If the attribute **userfqn** is set to *yes* on the source, the destination data set name is composed of the source current working directory concatenated with the source file name (the fully qualified file name).

Partitioned Data Sets

A file is considered partitioned if it has a data set organization of Partitioned Organization (PO) or a system managed type of Partitioned Data Set Extended (PDSE).

A file is referenced directly as a fully qualified name enclosed in apostrophes (') or as a relative name composed of one or more qualifiers concatenated to the current working directory value to form a fully qualified name. The qualifiers `.` and `..` that are used in the cd command do not have any special meaning in a file specification and will most likely result in an invalid fully-qualified data set name.

A partitioned data set member requires an additional member name as part of the file specification. The member name is enclosed within parenthesis as in APP.PDS(DATA1), where APP.PDS is the partitioned data set name and DATA1 is the member name.

Table 17.4 provides some examples of copy command destination file specifications for partitioned data sets. The examples assume that the fully qualified names are PDS's and a UDM logical name of SRV.

Current Directory	copy Command	Result
`TOM123`	`copy local=app1 srv=data`	`TOM123.DATA(APP1)`
`TOM123.DATA`	`copy local=app1`	`TOM123.DATA(APP1)`
`TOM123.DATA`	`copy local=app1 srv='GAM789.APP2.DATA'`	`GAM789.APP2.DATA(APP1)`
`TOM123.DATA`	`copy local=app1 srv=PDS(PR01)`	`TOM123.DATA.PDS(PR01)`

Table 17.4 copy Command Destination File Specifications for Partitioned Data Sets

Table 17.5 provides some examples of copy command source file specifications for partitioned data sets. The examples assume that the fully qualified names are PDS's and a UDM logical name of SRV.

Current Directory	copy Command	Result
`TOM123`	`copy srv=data(app1) local=app1.txt`	`TOM123.DATA(APP1)`
`TOM123.DATA`	`copy srv=app1 local=app1.txt`	`TOM123.DATA(APP1)`
`TOM123.DATA`	`copy srv='GAM789.DATA(APP1)' local=app1.txt`	`GAM789.DATA(APP1)`

Table 17.5 copy Command Source File Specifications for Partitioned Data Sets

Note:

- Member names are restricted to ISPF member naming conventions.
- The **`createop`** attribute values REPLACE and NEW apply to the member names and not to the partitioned data set.

DD File System

The semantics of a file specification is determined primarily by whether the ddname being referenced has a sequential or a partitioned data set allocated.

A ddname allocating a sequential data set is referred to as a sequential ddname, and a ddname allocating a partitioned data set is referred to as a partitioned ddname in the following text for purposes of brevity.

Sequential ddnames

A ddname is considered sequential if it allocates a data set with an organization of Physical Sequential (PS). A ddname reference is always a fully qualified name. A ddname must not be enclosed in apostrophes (').

There are three possible sequential ddname destination file specifications:

1. Name of the ddname defined by a JCL DD statement.
2. Current working directory value which is set to the name of a ddname and no destination file specification.
3. Source file specification if the current working directory value set to blanks and no destination file specification is provided.

There is one possible sequential ddname source file specification:

1. Name of the ddname defined by a JCL DD statement.

Table 17.6 provides some examples of copy command destination file specifications for sequential ddnames. The examples assume that ddname APPDD1 is defined with a JCL DD statement and a UDM logical name of SRV.

Current Directory	copy Command	Result
	`copy local=app1.txt srv=appdd1`	`APPDD1`
`APPDD1`	`copy local=app1.txt`	`APPDD1`
	`copy local=app1`	`APP1`

Table 17.6 copy Command Destination File Specifications for Sequential ddnames

Table 17.7 provides an example of copy command source file specifications for sequential ddnames. The examples assume that ddname APPDD1 is defined with a JCL DD statement and a UDM logical name of SRV.

Current Directory	copy Command	Result
	`copy srv=appdd1 local=app1.txt`	`APPDD1`

Table 17.7 copy Command Source File Specification for Sequential ddnames

Note: The **`createop`** attribute values REPLACE and NEW are not applicable to the sequential ddname file system.

Partitioned ddnames

A ddname is considered partitioned if it allocates a data set with an organization of Partition Organization (PO) or a system managed type of Partitioned Data Set Extended (PDSE). A ddname reference is always a fully-qualified name. A ddname must not be enclosed in apostrophes (').

A partitioned data set member requires an additional member name as part of the file specification. The member name is enclosed within parenthesis as in APPDD(DATA1), where APPDD is the ddname and DATA1 is the member name.

There are three possible partitioned ddname destination file specifications:

1. Name of the ddname defined by a JCL DD statement followed by a member name enclosed in parenthesis.
2. Current working directory value set to the name of a ddname defined by a JCL DD statement, and a member name specified as the destination file specification.
3. Current working directory value set to the name of a ddname defined by a JCL DD statement, and a member name specified by the source file specification if no destination file specification is provided.

There are two possible partitioned ddname source file specification:

1. Complete name of the ddname defined by a JCL DD statement followed by a member name enclosed in parenthesis.
2. Current working directory value set to the name of a ddname defined by a JCL DD statement and a member name specified as the source file name.

Table 17.8 provides some examples of **copy** command destination file specifications for partitioned ddnames. The examples assume that ddname APPDD1 is defined with a JCL DD statement and a UDM logical name of SRV.

Current Directory	copy Command	Result
	`copy local=app1.txt srv=appdd1(data1)`	`APPDD1(DATA1)`
`APPDD1`	`copy local=app1.txt srv=data1`	`APPDD1(DATA1)`
`APPDD1`	`copy local=app1`	`APPDD1(APP1)`

Table 17.8 copy Command Destination File Specifications for Partitioned ddnames

Table 17.9 provides an example of copy command source file specifications for sequential ddnames. The examples assume that ddname APPDD1 is defined with a JCL DD statement and a UDM logical name of SRV.

Current Directory	copy Command	Result
	`copy srv=appdd1(data1) local=app1.txt`	`APPDD1(DATA1)`
`APPDD1`	`copy srv=data1 local=app1.txt`	`APPDD1(DATA1)`

Table 17.9 copy Command Source File Specifications for Sequential ddnames

Note: The **`createop`** attribute values REPLACE and NEW are applicable to members of a partitioned ddname.

17.4 Copying Load Modules

UDM for z/OS provides the ability to copy load modules.

Note: Version 3.2.0 or greater of UDM must be used on both the source and destination sides of the transfer operation for a load module to be properly copied and usable.

The syntax for copying load modules is the same as any copy operation involving PDS/Es. However, there are some differences in how the copy operation takes place when the command to copy load module(s) is issued.

On the source side of the transfer, UDM uses **`IEBCOPY`** to unload the load modules, matching the source file mask from the PDS/E in which they reside into a temporary data set. It is this temporary data set that is transferred to the destination system. As a result, when UDM displays the status messages indicating that it is copying a file, it is the name of the temporary file that is displayed, since that is what is actually being transferred.

A temporary file name is also used on the destination side of the transfer. After the temporary file has been transferred, the load modules are ‘unpacked,’ using **`IEBCOPY`**, into a staging PDS/E (also using a temporary data set name). This PDS/E is created using the same attributes as the source PDS/E. From there, **`IEBCOPY`** is called a final time to move the load modules from the staging PDS/E to the final destination PDS/E. At this point, all of the temporary files are cleaned up.

The two-step process on the destination side of the transfer is used in case the blocking of the final destination data set does not match that of the source PDS/E.

17.4.1 Example

Figure 17.1, below, illustrates an example script of a load module transfer.

```
open src=* dst=dst-zos
attrib dst createop=new
copy src='MYHLQ.UDM.TESTLM(*)' dst='YOURHLQ.TEST.LOAD'
```

Figure 17.1 Load Module Transfer Script - Example

This simple script will copy all of the load modules from a PDS/E on the source system named **MYHLQ.UDM.TESTLM** to a newly created PDS/E on the destination system named **YOURHLQ.TEST.LOAD**.

Figure 17.2, below, illustrates an example of the output that is received when running a script such as that illustrated in Figure 17.1.

```
Data session established using cipher: NULL-MD5
Two-party session established with 2 (component 1208550125)
Transfer mode settings:
type=binary
trim=no
Session options:
Keep Alive Interval: 120
Network Fault Tolerant: yes
src: Packaging up the following files in
     'MYHLQ.UDMTMP.STC07047.R2EED53.N0000000'
src:  LM1
src:  LM2
src:  tmp
src: 'MYHLQ.UDMTMP.STC07047.R2EED53.N0000000' is being transferred in binary
      mode
src: 'YOURHLQ.UDMTMP.JOB07063.RD36420.N0000000' will be used as the
      destination filname
dst:  Unpacking from 'YOURHLQ.UDMTMP.JOB07063.RD36420.N0000000'
src: 'MYHLQ.UDMTMP.STC07047.R2EED53.N0000000' transfered successfully in
      0:01:55.
src:  10566891 bytes read    10566891 bytes written
src:  Transfer operation complete. 1 file(s) copied in 0:01:55.448.
src:  10566891 bytes transferred (91529.44 bytes per second)
```

Figure 17.2 Load Module Transfer Script - Output

At the beginning of the copy operation, the source side indicates that it is packaging the load modules into a temporary data set, **`MYHLQ.UDMTMP.STC07047.R2EED53.N0000000`**. This is the temporary data set that is transferred using a data set name of **`YOURHLQ.UDMTMP.JOB07063.RD36420.N0000000`** on the destination side. It is from **`YOURHLQ.UDMTMP.JOB07063.RD36420.N0000000`** that the load modules are unpacked into the temporary staging data set before being moved into the final destination PDS/E, **`YOURHLQ.TEST.LOAD`**, which was specified in the copy command.

As you can see, some of the output from a copy operation involving load modules may vary from the output when copying other types of data sets. However, the nomenclature of the copy command has not changed.

Likewise, attributes such as **`CREATEOP`**, **`DIRBLOCKS`**, and others work the same way with load modules as they do with other types of data sets. This includes the caveat that the attribute settings must be compatible with the type of data set(s) involved in the transfer.

17.4.2 Error Reporting

It is possible for the **`IEBCOPY`** portions of a load module transfer to fail. If this occurs, UDM prints the output from **`IEBCOPY`** in the transaction log.

17.4.3 Special Attributes

UDM uses heuristics in determining the space attributes for allocating the temporary data sets. The volume that these data sets reside is chosen by the system.

The **`TMPVOLSER`** attribute lets you set the volume on which the temporary files will be allocated.

- Setting this attribute on the source side specifies the location of the temporary sequential data set that will be transferred.
- Setting this attribute on the destination side specifies the volume for the temporary transfer file as well as the volume used by the temporary staging PDS/E.

The **`TMPPRIMSPACE`**, **`TMPSECSPACE`**, and **`TMPSPACEUNIT`** attributes specify the amount of primary space and secondary space used when allocating the temporary files as well as the space unit used.

- When set on the source side, these attributes affect the temporary sequential data set that will be transferred.
- When set on the destination side, these attributes are used in allocating the temporary transfer file and the temporary staging PDS/E.

The **`TMPDIRBLOCKS`** attribute is used only on the destination side. It specifies the number of directory blocks used by the staging PDS/E.

Note: Although you can override these attributes, it is not recommended.

CHAPTER 18

Transfer Operations (IBM i-Specific)

18.1 Overview

This section describes information that is specific to IBM i file transfer operations:

- IBM i I/O
- Codepage - CCSID mappings
- UDM commands under IBM i

18.2 IBM i I/O

This section describes the file systems and file types available in UDM for IBM i.

18.2.1 File Systems

UDM for IBM i supports two types of file systems:

1. LIB (Library) file system
2. HFS (IFS: root and QOpenSys)

The default file system for UDM on the IBM i is LIB.

18.2.2 HFS (for IBM i) File System

HFS follows the CFS rules in Section 16.4 UDM Common File System.

It supports stream files under the root and QOpenSys IFS file systems. Users using UDM to access file systems under IFS, other than root and QOpenSys, do so at their own risk.

HFS also provides enhanced **`eol`** handling and **`eol`** attribute values for the mixed ASCII and EBCIDIC environment. See Section 16.7.3 End of Line Sequence for details.

18.2.3 LIB File System

LIB follows the extensions to CFS outlined in Section 8.5.2 LIB.

File Types

UDM for IBM i supports three file types of the LIB file system:

1. Data Physical Files
2. Source Physical Files
3. Save Files

The type of file created in a copy command on the destination side is governed by the IBM i-specific FILETYPE attribute (see Table 18.2 IBM i-Specific LIB File Attributes for Creating New Files).

The default file type created in a copy command is a data physical file.

18.2.4 Data Physical Files Support

UDM for IBM i supports data physical files with a CCSID and with no DDS (default CCSID of 65535).

If a DDS is attached to a data physical file on the source side, that same DDS is used when doing an IBM i to IBM i copy on the destination side unless the DDSFILE, DDSLIB, and/or DDSMBR attributes are overridden on the destination side to indicate a different DDS or no DDS is to be used.

The DDS itself is not copied, so it must reside on the destination side.

There is one exception: if the source side is a file created via FTP, the created file has an associated DDS file. The associated DDS specifies a single field and DDS source identified by the file is deleted following completion of the job under which the file was created. When UDM identifies a file created by FTP, it ignores the DDS and copies the file as though no DDS exists.

When copying any file to a destination data physical file with the DDSFILE, DDSLIB, and DDSMBR attributes set to point to the file, library, and member of an existing DDS, that DDS is attached to the destination file.

In either case, whether from the source or explicitly on the destination side, if a DDS is used on the destination side, the resulting file's CCSID is determined by the DDS or by the job CCSID settings if not provided by the DDS.

If the source file has no DDS, or if the destination attributes specify no DDS (or are overridden to do so to prevent the source attributes used in an IBM i to IBM i copy), the destination data physical file is created with a CCSID of 65535 (meaning no translation).

UDM will issue an informational message if you try to transfer a source file that has a DDS in text mode that tells the user corruption is likely. This is because text translation on the field level is governed by the DDS. UDM does not support independent field-level text translation.

Caution about Text Mode Transfer of Files with DDS

In general, files with DDS should be transferred using binary mode only.

There are instances when a user may want to use text mode. However, without an advanced user's thorough understanding of CCSID and code page, unexpected results will occur.

As of Stonebranch Solutions for IBM i, release 3.2.0, when the correct conditions are met, UDM maps the code page attribute associated with the data stream to a CCSID. This occurs only when data is transferred to a data physical file in text mode with an associated DDS file.

This mapping is used on the LIB file open to obtain translation between the data stream and data in fields with CCSIDs other than 65535. The translation is done by IBM i; UDM is in no way involved with this translation process.

18.2.5 Source Physical Files Support

Source physical files have a common, known DDS. This DDS specifies the following record format:

- First six bytes contain a sequence number
- Next six bytes contain a line modification date
- Remaining number of bytes are text data This length can be between 1 and 32754 bytes for single-byte character systems.

A single attribute, USESRCSEQ (with values of YES or NO) governs whether or not the sequence number and modification date are included in the source record when transferring a source physical file. How this happens depends on the mode type of the transfer. By default, this value is set to NO, meaning sequence numbers and modification dates are to be stripped.

When writing a source physical file, the USESRCSEQ attribute specifies whether or not source sequence information is expected to be included in the source data. If the value is set to NO, UDM generates sequence number and modification date information. Otherwise, the first 12 bytes of each source record contain that information. This value is sent as a source attribute in IBM i to IBM i copies, so unless it is overridden, it automatically will tell the destination side if the sequence numbers are in the data. Allowing this option to be set permits the effective copying of source physical files from non-400 systems that already contain sequence number information.

When creating UDM sequence data, two additional destination side attributes are used.

- SEQSTART specifies the starting sequence number of the first record written and range from 0000.01 to 9999.99. The default is 0001.00.
- SEQINCR indicates how much the sequence number is incremented from record to record. Valid values are 00.01 to 99.99. The default value is 01.00.

SEQSTART and SEQINCR are sent as source attributes, but used on the destination side only when a new file is being created. If UDM is replacing or appending to an existing source physical file, the values of SEQSTART and SEQINCR of the existing destination file are used.

18.2.6 Copying Source Physical Files

Source physical files can be copied in both text and binary mode.

There are three possible copy operations involving source physical files:

1. Like Copies of Source Physical File Data
2. Non-Source Physical to Source Physical Copies
3. Source Physical to Non-Source Physical Copies

Like Copies of Source Physical File Data

IBM i to IBM i copy: both the source and destination are source physical files.

In this case, if the sequence and date fields are not stripped from the source, this data is written as is to the destination. If the fields are stripped, the SEQSTART and SEQINCR attributes define how the sequence data is generated (described later in this section) and the current date is placed in the date field.

Non-Source Physical to Source Physical Copies

Non-source physical file is the source and a source physical file is the destination.

In this case, the SECSTART and SEQINCR attributes are used to create the sequence number and the date field is seeding with the current date.

Source Physical to Non-Source Physical Copies

Source records are read and formatted as described above and the destination system writes them out as dictated by its attributes.

18.2.7 Save Files Support

Save (SAVF) files are essentially binary archives of data. They may contain one or more objects inside. These objects can be extracted individually or in their entirety.

UDM for IBM i supports NEW and REPLACE operations for CREATEOP when a Save file is the destination file type. It does not support append operations on Save files.

SAVF to SAVF Transfers

Copying a Save file to a Save file always should be performed via a binary transfer, regardless of the mode type setting. All of the source data is read in binary and written in binary. This type of transfer succeeds only if the CREATEOP is set to NEW or REPLACE.

If the destination Save file already exists, and the CREATEOP is set to APPEND, UDM issues an error and aborts the transfer.

Non-SAVF to SAVF Transfers

As with all cases when a Save file is involved, a binary transfer should be forced. The source data is written to the destination in the form it is read. Only the values of NEW and REPLACE are supported for CREATEOP on the destination side when the destination file type is a Save file.

Note: The non-Save file must be a file that originally was created as a Save file on an IBM i system and then stored as a binary file on a non-IBM i system.

SAVF to Non-SAVF Transfers

Data is read and transferred automatically in binary to the destination machine.

18.2.8 File Specifications

File specifications in the LIB file system consist of up to three components - library, file, and member - that take the following form:

```
LIBRARY/FILE(MEMBER)
```

Note: Data physical files and source physical files have members. Save files do not.

18.2.9 Wild Cards

In source file specifications in the LIB file system for the delete command and for the copy command and file specifications for the `forfiles` statement, wildcards can appear in the library, file, and/or member portions. An asterisk (*****) represents a match of zero or more characters. A question mark (**?**) represents a match of exactly one character,

Wildcards only apply to the library, file or member portion of the fully qualified file name in which they appear. For example, in the statement COPY SRC=ABC/DEF*, the wildcard only applies to the file portion of the name and an error will result because the user did not provide a member name. To copy all of the files that begin with DEF, along with all of their members from library ABC, use the format COPY SRC=ABC/DEF*(*). Likewise, to copy all of the files and their members in libraries that begin with ABC, use the format COPY SRC=ABC*/*(*).

In destination file specifications, wild cards are not allowed.

Examples

```
COPY SRC=ABC*/DEF*
```

Copies all files beginning with DEF from all libraries beginning with ABC.

```
COPY SRC=ABC/DEF(*)
```

Copies all the members in the physical file DEF in the library ABC.

```
DELETE SRC=MYLIB/MYFILE?
```

Deletes all files in the library MYLIB starting with MYFILE and containing one additional character.

```
FORFILES SRC=*/*(*)
```

Lists all members in all files in all libraries.

18.3 Codepage - CCSID Mappings

Information that is stored, moved, and displayed on IBM i has a CCSID (Coded Character Set IDentifier) number associated with it. UDM uses these CCSID numbers, where appropriate, when creating data files, transferring data, and storing data.

Each language available on IBM i has an associated CCSID. The CCSID identifies the mapping of numeric representations associated with each letter or symbol represented by the computer. It also identifies the glyphs required to represent those characters and symbols when displayed. UDM is not concerned about the display aspect of a CCSID or the associated data, only about the mapping between these numeric representations.

Code pages provide one mechanism of mapping (translating) between these numeric representations. Another means of representing these mappings is to use two CCSIDs: one for the data origin and another for the data destination. For example, when writing data to a file on IBM i, the data stream being sent to the file has an associated CCSID and the file itself has an associated CCSID. In this way, the operating system knows how to provide the translation between data to be written to a file and the data that is physically on the disk file.

IBM i

For UDM on IBM i, the data stream CCSID is established via the code page to CCSID mapping file which is controlled by the UDM Manager CODEPAGE_TO_CCSID_MAP configuration file option and the UDM Server CODEPAGE_TO_CCSID_MAP configuration file option.

Of course, the translation also works the other way around, when data is read from a disk it is translated from the physical disk back to the data stream. One special CCSID is 65535, which indicates that no translation is to take place.

When transferring data between computer systems UDM allows the specification of a code page for each system. For example:

```
open source=winsys45 user=id1 pwd=mypwd codepage=iso8859-1
destination=os400trex user=id2 pwd=newpwd codepage=IBM037
```

This tells UDM that the two code pages iso8859-1 and IBM037 are to be used for mapping data between the two systems.

Very often, the numeric portion of a code page also is a CCSID to which the code page relates. In this case, the numeric representations represented by the code page are the same as those represented by the CCSID. One example of this common identification is the code page IBM037 and the CCSID 037. This code page and CCSID represent the native numeric representation of data under IBM i.

The default code page for UDM is IBM037. This is the internal code page, as well as the default external code page used for the control session and data session, unless overridden by the configuration file or the CODEPAGE parameter on the open command (data session only).

18.3.1 CCSID Mapping

In order to get data to and from a file with a given CCSID, a corresponding CCSID matching the data session code page must be used in order to map the data correctly.

The data stream CCSID is mapped from the code page via the code page to CCSID mapping table. By default, internal tables provide this mapping; however, see the UDM Manager CODEPAGE_TO_CCSID_MAP configuration option or the UDM Server CODEPAGE_TO_CCSID_MAP configuration option in the UDM Reference Guide regarding setting up an external file.

If a mapping cannot be made, the following occurs:

1. Warning is issued to the user.
2. Copy operation fails.

ASCII code pages can map to CCSIDs that are available in the HFS file system but not the LIB file system. If one of these code pages is used, a different warning should be issued that lets the user know that the mapping will work for HFS, but the behavior in the LIB file system is indeterminate.

Table 18.1, below, contains those mappings.

Note: If a code page contains a dash (**-**) in the name (for example, ISO8859-1), an underscore (**_**) must replace the (**-**) when the code page is used in a UDM script.

Code Page	CCSID	HFS	LIB
IBM037	037	√	√
IBM273	273	√	√
IBM277	277	√	√
IBM278	278	√	√
IBM280	280	√	√
IBM284	284	√	√
IBM500	500	√	√
IBM1047			
IBM1140	1140	√	√
IBM1141	1141	√	√
IBM1142	1142	√	√
IBM1143	1143	√	√
IBM1144	1144	√	√
IBM1145	1145	√	√
IBM1146	1146	√	√
IBM1147	1147	√	√
IBM1148	1148	√	√

Code Page	CCSID	HFS	LIB
ISO8859-1	819	√	
ISO8859-2	912	√	
ISO8859-4	914	√	
ISO8859-5	915	√	
ISO8859-6	1089	√	
ISO8859-7	813	√	
ISO8859-8	916	√	
ISO8859-9	920	√	
ISO8859-13	921	√	
ISO8859-15	923	√	
PC437	437	√	
PC737	737	√	
PC775	775	√	
PC850	850	√	
PC852	852	√	
PC855	855	√	
PC857	857	√	
PC860	860	√	
PC861	861	√	
PC862	862	√	
PC863	863	√	
PC864	864	√	
PC865	865	√	
PC866	866	√	
PC869	869	√	
PC874	874	√	
WIN1250	1250	√	
WIN1251	1251	√	
WIN1252	1252	√	
WIN1253	1253	√	
WIN1254	1254	√	
WIN1255	1255	√	
WIN1256	1256	√	
WIN1257	1257	√	
WIN1258	1258	√	

Table 18.1 CCSID Mappings

18.4 Command Reference

This section describes UDM command behavior when working with the LIB and HFS file systems.

18.4.1 attrib (Attribute) Command

UDM provides three attribute levels. In order of precedence, from lowest to highest, they are:

1. Default (lowest priority)
2. Source
3. Override (highest priority)

When a user sets an attribute, the override attribute level is being set. Default attributes are those set by UDM at startup. Source attributes are attributes that UDM obtains from the source file and uses for the destination file. For example, when transferring a file one IBM i LIB location to another, UDM reads the record length of the source file and uses the source file record length to create the destination file. If the source file is on UNIX or in the HFS file system, record length has no meaning and the source attribute is not set.

In addition to the standard UDM file attributes (CREATEOP, EOL, LINELEN, LINEOP, PADLINE, and TRUNCEXT), IBM i-specific file attributes are required in order to create new files in the LIB and HFS file systems. However, not all attributes are required for all file types. (For information on which attributes can be used with each file type, refer to IBM i online documentation.)

File Attributes

The following file attributes tables (Table 18.2 and Table 18.3) provides the following information about IBM i-specific file attributes and the file types for which they are required:

Name
Name of the attribute to be used in the LIB file system

Description
Description of the attribute

Source
Indication of whether or not the attribute is a source attribute. A source attribute is one whose destination side value is taken from its source side unless the user explicitly has overridden the destination side value. Source attributes are used only when both of the these conditions apply:

- Copying from one IBM i system to another
- Both source and destination are in the LIB file systems

Value
Values that can be assigned to the attribute

UDM Default
UDM default value that is assigned to an attribute if a value is not otherwise assigned. A UDM default of NULL identifies the attribute as available to be set, but not set initially. If the attribute value is NULL (or empty string), the system default is used.

System Default
IBM i system default value that is assigned to an attribute if its UDM default is NULL or empty string. An attribute can have different system defaults for different file types.

File Type
Types of files in the LIB and HFS file systems to which an attribute applies:

- LIB Library
- PF Data physical file
- SP Source physical file
- SAVFSave file
- Stream

LIB File System Attributes

Table 18.2 identifies attributes that are unique to the LIB file system.

Name	Description	Source	Value	UDM Default	System Default	File Type
ACCPTH	Access path type		ARRIVAL, KEYED	NULL	ARRIVAL	SP
ALWDLT	Allow delete operation	√	YES, NO	NULL	YES	PF, SP
ALWUPD	Allow update operation	√	YES, NO	NULL	YES	PF, SP
ASPDEV	ASP device		ASP, ASPGRPPRI, SYSTEM, device name	NULL	ASP	LIB
ASPNUM	ASP number		LIBASP, 1-32, ASPDEV	NULL	LIBASP for SAVF, 1 for LIB	LIB, SAVF
AUT	Authority		LIBCRTAUT, ALL, CHANGE, EXCLUDE, USE [2]	NULL	LIBCRTAUT for LIB,PF,SP EXCLUDE for SAVF	LIB, PF, SP, SAVF
CCSID *	CCSID of the file (source physical files only). For data physical files, the DDS (if one is given) determines its CCSID; if no DDS is given, the value is 65535.	√	EBCDIC CCSIDs	CODEPAGE		SP

Name	Description	Source	Value	UDM Default	System Default	File Type
CRTAUT	Create authority		SYSVAL, ALL, CHANGE, EXCLUDE, USE, authority name	NULL		LIB
DDSLIB	Library of the DDS used to describe the file	√		empty string		PF
DDSFILE	File of the DDS used to describe the file	√		empty string		PF
DDSMBR	Member of the DDS used to describe the file	√		empty string		PF
DLTPCT	Maximum percentage of deleted records allowed		1-100, NONE	NULL	NONE	PF
EXPDATE	Expiration date for member	√	date, NONE	NULL	NONE	PF, SP
FILETYPE	Type of file to create when creating a new file	√	DATA, SRC, SAVF	DATA		PF, SP, SAVF
FRCRATIO	Records to a force write		integer, NONE	NULL	NONE	PF, SP
GENLVL	Generation severity level		0-30	NULL	20	PF
LIBTYPE	Type of library created when creating a library	√	PROD, TEST	PROD		LIB
LVLCHK	Record format level check	√	YES, NO	NULL	YES	PF
MAXMBRS	Maximum number of members	√	integer, NOMAX	NULL	1 for PF, NOMAX for SP	PF, SP
MAXRCDS	Maximum number of records		1-2146762800, NOMAX	NULL		SAVF
OPTION	Source listing options		SRC, NOSRC, SOURCE, NOSOURCE, LIST, NOLIST, SECLVL, NOSECLVL, EVENTF, NOEVENTF (up to four repetitions)	empty string		PF
RCDLEN	Record length if no DDS is used	√	integer	92		PF, SP
REUSEDLT	Reuse deleted records	√	YES, NO	NULL	NO	PF
SEQSTART	Beginning sequence number used when writing to a source physical file	√	0000.01 – 9999.99	1.00		SP
SEQINCR	Amount to increment sequence number by when writing a record to a source physical file	√	00.01 – 99..99	1.00		SP
SHARE	Share open data path	√	YES, NO	NULL	NO	PF, SP, SAVF

Name	Description	Source	Value	UDM Default	System Default	File Type
SIZE	Member size	√	• Single values: NOMAX • Other values: Comma-separated element list • Element 1: Initial number of records 1-2147483646, • Element 2: Increment number of records Integer, • Element 3: Maximum increments Integer (EX: 10000,1000,3)	size_attrib configuration file entry if provided; otherwise empty string	10000,1000, 3 for PF, 10000,1000, 499 for SP	PF, SP
USESRCSEQ	Sequence number and modification date information: • On Source side: retain this information when copying a source physical file • On Destination side: Record data includes this information	√	YES, NO	NO		SP
WAITFILE	Maximum file wait time		integer, IMMED, CLS	NULL	30 for PF IMMED for SP, SAVF	PF, SP, SAVF
WAITRCD	Maximum record wait time		integer, IMMED, NOMAX	NULL	60	PF, SP
* With CCSID set to CODEPAGE, when the UDM CCSID attribute is not set either explicitly or implicitly via an IBM i to IBM i file transfer, the CCSID associated with the code page via the code page to CCSID mapping tables gets used as the CCSID attribute value. One implication is that, by default, files may be created with the CCSID associated with the codepage option.						

Table 18.2 IBM i-Specific LIB File Attributes for Creating New Files

HFS Attributes

Table 18.3 identifies attributes that are unique to the HFS file system. (Currently, there is only one HFS unique attribute, CCSID.)

Name	Description	Source	Value	UDM Default	System Default	File Type
CCSID	CCSID of the file	√	EBCDIC and ASCII CCSIDs	CODEPAGE		stream

Table 18.3 IBM i -Specific HFS File Attributes for Creating New Files

18.4.2 call (Call) Command

To invoke a script, the member name is required and can be **`*FILE`**:

```
call mylib/myfile(myscript)
```

Specifying *FILE invokes the normal default IBM i file search order.

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

The following example illustrates both:

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

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

18.4.3 cd (Change Directory) Command

When you authenticate with a UDM Server running under IBM i, the current library is set to the default library for that user.

In file operations where the library is not identified explicitly as a part of the file specification, the current library is used instead.

Example:

```
COPY SRC=C:\MYFILE DST=MYFILE(MYMEMB)
```

With a current library set to MYUSER, this command will result in a destination file specification name of MYUSER/MYFILE(MYMEMB).

You can change the current library by issuing the cd (Change Directory) command with the new library name as in this example:

```
CD DST=YOURUSER
```

There is a special case, when using UDM from one IBM i machine to another, where the source library name can be used instead (see Section 14 Samples for more details). In order for this to work, you must first clear the destination current library by issuing the following command:

```
CD DST=..
```

18.4.4 copy (Copy) Command

In both the HFS and LIB file systems, if a file with multi-byte characters, including DBCS (Double Byte Character Set), is transferred using UDM in text mode, data loss or corruption can occur. This is because UDM is basically SBCS (Single Byte Character Set) in nature.

If an SBCS code page is used for the data transfer in text mode, some data can be translated into characters that do not translate back to the same data when written to the target file.

To transfer these type of files, users normally should use binary mode and should be very careful if they find it necessary to use text mode.

18.4.5 File Specification Rules

File specifications can appear in a variety of UDM commands, from `copy` to `forfiles`.

On IBM i, a simple set of rules governs how the full file specification used in an operation is constructed.

Since there are still subtle differences between source and destination side file specifications, in terms of how they are derived, separate rules are provided for each type of specification:

- Source File Specification
- Destination File Specification
  - Source and Destination in LIB
  - Destination (only) in LIB

Source File Specification Rules

The following rules apply to file specifications that are in the source position in a **copy** command.

(In all examples, CURLIB is the current library.)

1. If the file specification contains only the file portion, the current library is pre-pended to the name to refer directly to a file with no member component.

   Example:

   ```
   COPY SRC=MYFILE
   ```
   The absolute path derived would be CURLIB/MYFILE.
2. If the file specification contains only file and member portions, the current library is pre-pended to the name to refer to a specific member in a file.

   Example:

   ```
   COPY SRC=MYFILE(MYMBR)
   ```
   The absolute path derived would be CURLIB/MYFILE(MYMBR).
3. If the file specification contains only library and file portions, an absolute path without a member component is used.

   Example:

   ```
   COPY SRC=MYLIB/MYFILE
   ```
   The absolute path would be exactly as given: MYLIB/MYFILE.
4. If a file specification contains library, file, and member portions, all of those components are used explicitly in the absolute path.

   Example:

   ```
   COPY SRC=MYLIB/MYFILE(MYMBR)
   ```
   The absolute path would be MYLIB/MYFILE(MYMBR).

Destination File Specification Rules

Destination path names follow many of the same rules as source path names, with one big exception: all or part of the destination path name may be derived using a name (or names, in the case of IBM i to IBM i LIB file system copies) coming from the source side of a transfer operation.

Source and Destination in LIB File System

The following rules apply for IBM i to IBM i transfers where both the source and destination are operating in the LIB file system.

In these example, the current destination library is DSTLIB and the absolute path of the source file being copied is MYLIB/MYFILE(MYMBR).

1. If the destination file specification contains an empty path (no library, file, or member portions), the file and member portions are derived from the source path. If the destination file is to be a save file, the absolute path in this case would be DSTLIB/MYFILE. If the destination file is to be a physical file, the absolute path would be DSTLIB/MYFILE(MYMBR).

   Examples:

   `COPY SRC=MYLIB/MYFILE(MYMBR)`
   The result is a destination name of DSTLIB/MYFILE(MYMBR) if the destination file type is a physical file.

   `COPY SRC=MYLIB/MYFILE`
   The result is a destination name of DSTLIB/MYFILE if the destination file type is a save file.

2. If the destination file specification contains only a file portion, the current library is pre-pended to the absolute path. In this case, if the destination file is to be a save file, the absolute path would be DSTLIB/YOURFILE. If the destination file is to be a physical file, the absolute path would be DSTLIB/YOURFILE(MYMBR).

   Examples:

   `COPY SRC=MYLIB/MYFILE(MYMBR) DST=YOURFILE`
   The result is a destination name of DSTLIB/YOURFILE(MYMBR) if the destination file type is a physical file.

   `COPY SRC=MYLIB/MYFILE DST=YOURFILE`
   The result is a destination name of DSTLIB/YOURFILE if the destination file type is a save file.

3. If the destination file specification contains only a file portion (with an empty member), the result is exactly the same as when just a destination file name is given.

   Example:

   `COPY SRC=MYLIB/MYFILE(MYMBR) DST=YOURFILE()`
   The result is a destination name of DSTLIB/YOURFILE(MYMBR) if the destination file type is a physical file.

4. If the destination file specification contains only file and member portions, the resulting absolute path is DSTLIB/YOURFILE(YOURMBR) if a physical file is wanted.

   Example:

   ```
   COPY SRC=MYLIB/MYFILE(MYMBR) DST=YOUFILE(YOURMBR)
   ```
   The result is a destination name of DSTLIB/YOURFILE(YOURMBR) if the destination file type is a physical file.

5. If the destination file specification contains only a library portion, that library is used instead of the current library. In this case, an absolute path of YOURLIB/MYFILE is used if a save file is wanted. If a physical file is wanted, an absolute path of YOURLIB/MYFILE(MYMBR) is used.

   Examples:

   ```
   COPY SRC=MYLIB/MYFILE(MYMBR) DST=YOURLIB/
   ```
   The result is a destination name of YOURLIB/MYFILE(MYMBR) if the destination file type is a physical file.

   ```
   COPY SRC=MYLIB/MYFILE DST=YOURLIB/
   ```
   The result is a destination name of YOURLIB/MYFILE if the destination file type is a save file.

6. If the destination file specification contains only library and file portions, an absolute path of YOURLIB/YOURFILE is derived if a save file is wanted. If a physical file is wanted, YOURLIB/YOURFILE(MYMBR) is used.

   Examples:

   ```
   COPY SRC=MYLIB/MYFILE(MYMBR) DST=YOURLIB/YOURFILE
   ```
   The result is a destination name of YOURLIB/YOURFILE(MYMBR) if the destination file type is a physical file.

   ```
   COPY SRC=MYLIB/MYFILE DST=YOURLIB/YOURFILE
   ```
   The result is a destination name of YOURLIB/YOURFILE if the destination file type is a save file.

7. If the destination file specification contains library and file portions, as well as an empty member name, the result is exactly the same as when the file specification contains only library and file portions.

   Example:

   ```
   COPY SRC=MYLIB/MYFILE(MYMBR) DST=YOURLIB/YOURFILE()
   ```
   The result is a destination name of YOURLIB/YOURFILE(MYMBR) if the destination file type is a physical file.

8. If the destination file specification contains a complete absolute path (library, file, and member portions), the source file name has no effect on the destination path in any way. In this case, if the destination file type is a save file, YOURLIB/YOURFILE is used. If the destination file type is a physical file, YOURLIB/YOURFILE(YOURMBR) is used.

   Examples:

   `COPY SRC=MYLIB/MYFILE(MYMBR)DST=YOURLIB/YOURFILE(YOURMBR)`
   The result is a destination name of YOURLIB/YOURFILE(YOURMBR) if the destination file type is a physical file.

   `COPY SRC=MYLIB/MYFILE DST=YOURLIB/YOURFILE`
   The result is a destination name of YOURLIB/YOURFILE if the destination file type is a save file.

9. In cases where a member is specified explicitly in the destination file name and the destination file type is a save file, an error is issued.

   Note: If the user issues a **`cd dst-logical-name=..`** command to blank out the current library on the destination side, the library name in the absolute path of the source file is used in the destination absolute path in cases where no library is specified explicitly.

   This works only for IBM i to IBM i copies where both operating systems are operating in the LIB file system.

   Example:

   `CD DST=..`
   `COPY SRC=MYLIB/MYFILE(MYMBR)`
   The result is a destination of MYLIB/MYFILE(MYMBR), using the source's library, file, and member names, because none are supplied explicitly in the **`copy`** command. The current directory on the destination side is empty because the command **`cd DST=..`** was issued.

Destination (only) in LIB File System

Transfers where only the destination is operating in the LIB file system produce slightly different results.

The following rules apply for:

- Copies from non-IBM i machines to an IBM i machine operating in the LIB file system
- Copies from IBM i machines working in the HFS file system to an IBM i machine operating in the LIB file system,

These operations do not make use of source attributes describing all the library and file portions of the source file specification.

(in the following example, the source file being copied is MYFILE and the current library on the destination side is MYDSTLIB.)

1. If the destination file specification contains an empty path (no library, file, or member portions), the source file name is used for the file and member names on the destination side. If the destination file is to be a save file, the absolute path in this case would be DSTLIB/MYFILE. If the destination file is to be a physical file, the absolute path would be DSTLIB/MYFILE(MYFILE).

   Example:

   ```
   COPY SRC=MYFILE
   ```
   DSTLIB/MYFILE(MYFILE) will be used as the destination name if the destination file type is a physical file and DSTLIB/MYFILE will be used if the destination file type is a save file.
2. If the destination file specification contains only a file portion, the current library is pre-pended to the absolute path and the source file name is used for the member (if it applies). In this case, if the destination file is to be a save file, the absolute path would be DSTLIB/YOURFILE. If the destination file is to be a physical file, the absolute path would be DSTLIB/YOURFILE(MYFILE).

   Example:

   ```
   COPY SRC=MYFILE DST=YOURFILE
   ```
   DSTLIB/YOURFILE(MYFILE) will be used as the destination name if the destination file type is a physical file and DSTLIB/YOURFILE will be used if the destination file type is a save file.
3. If the destination file specification contains only a file portion and an empty member portion, the result is exactly the same as when the file specification contains only file portion.

   Example:

   ```
   COPY SRC=MYFILE DST=YOURFILE()
   ```
   DSTLIB/YOURFILE(MYFILE) will be used as the destination name if the destination file type is a physical file.

4. If the destination file specification contains only file and member portions, the resulting absolute path is DSTLIB/YOURFILE(YOURMBR) if a physical file is wanted.

   Example:

   ```
   COPY SRC=MYFILE DST=YOURFILE(YOURMBR)
   ```
   DSTLIB/YOURFILE(YOURMBR) will be used as the destination name if the destination file type is a physical file.

5. If the destination file specification contains only a library portion, that library is used instead of the current library. In this case, an absolute path of YOURLIB/MYFILE is used if a save file is wanted. If a physical file is wanted, an absolute path of YOURLIB/MYFILE(MYFILE) is used.

   Example:

   ```
   COPY SRC=MYFILE DST=YOURLIB/
   ```
   YOURLIB/MYFILE(MYFILE) is used as the destination name if the destination file type is a physical file or YOURLIB/MYFILE if the destination file type is a save file.

6. If the destination file specification contains only library and file portions, an absolute path of YOURLIB/YOURFILE is derived if a save file is wanted. If an absolute path of YOURLIB/YOURFILE(MYFILE) is used if a physical file is wanted,

   Example:

   ```
   COPY SRC=MYFILE DST=YOURLIB/YOURFILE
   ```
   YOURLIB/YOURFILE(MYFILE) is the destination name if a physical file is wanted and YOURLIB/YOURFILE is used if a save file is wanted.

7. If the destination file specification contains library and file portions, as well as an empty member portion, the result is exactly the same as when the specification contains only a library and file portions.

   Example:

   ```
   COPY SRC=MYFILE DST=YOURLIB/YOURFILE
   ```
   YOURLIB/YOURFILE(MYFILE) is the destination name if a physical file is wanted.

8. If the destination file specification contains a complete absolute path (library, file, and member portions), the source file name has no effect on the destination path in any way. In this case, if the destination file type is a physical file, YOURLIB/YOURFILE(YOURMBR) is used.

   Example:

   ```
   COPY SRC=MYFILE DST=YOURLIB/YOURFILE(YOURMBR)
   ```
   YOURLIB/YOURFILE(YOURMBR) is the destination if the destination file type is a physical file.

9. In cases where a member is specified explicitly in the destination file name and the destination file type is a save file, an error is issued.

18.4.6 delete (Delete) Command

The delete (Delete) command in the UDM for IBM i LIB file system takes the following form:

```
DELETE logical-name=file-mask
```

delete Command Requirements

The delete command has the following requirements:

- It can be used to remove files and members, but not libraries.

Note: For the protection of the file system, UDM for IBM i does not allow users to delete libraries.

delete Command Forms

With UDM for IBM i, the file mask, which can contain wild cards in any portion (library, file and member) takes one of the following forms.

Name	Description
DELETE logical-name=LIBRARY/FILE	Deletes any files (including Save files) in the libraries that match the mask.
DELETE logical-name=FILE	Deletes any files (including Save files) in the current directory (library) that match the mask.
DELETE logical-name=LIBRARY/FILE(MEMBER)	Deletes any members where the library, file, and member portions of their fully qualified names match the appropriate elements of the mask
DELETE logical-name=FILE(MEMBER)	Deletes any members in the current directory whose file and member portions of their fully qualified names match the appropriate elements of the mask

Table 18.4 delete Command Forms with UDM under IBM i

18.4.7 rename (Rename) Command

The **rename** (Rename) command in the UDM for IBM i LIB file system takes the following form.

```
 RENAME logical-name old-name new-name
```

rename Command Requirements

The **rename** command has the following requirements:

- Libraries cannot be renamed.
- A single object level (file or member) can be renamed only with a single call. The name of a file and one of its members cannot be renamed with a single call. All other cases result in a failure.
- Wild cards are not allowed.
- It can be used only at the file and member level; it cannot be used to rename libraries, However, rename can be used to move existing files to existing libraries.
- It cannot be used to move a member from one file to another, since the destination file may not have the same attributes (for example, record length) as the source file. This could result in corrupt (or seemingly corrupt) data.
- It cannot be used to move a file from one library to another because it should not be used to create new libraries.

rename Command Forms

Name	Description
RENAME logical-name LIBRARY/FILE LIBRARY/FILE	Renames the file in the old portion with file name in the new portion. The library in the new name portion of the rename must match the library name in the old name portion.
RENAME logical-name FILE FILE	Renames the file in the current library in the old portion with the new file name in the current directory (library).
RENAME logical-name FILE LIBRARY/FILE	Renames the file in the current directory with the file name in the new portion. The library in the old name portion must be the current library.
RENAME logical-name LIBRARY/FILE FILE	Renames the file in the given library with the name of the file in the new portion, if the library name in the old portion is the same as the current library
RENAME logical-name LIBRARY/FILE(MEMBER) LIBRARY/FILE(MEMBER)	Renames the old member name with the new member name. Both the library and file name portions in the old and new member names must match
RENAME logical-name FILE(MEMBER) FILE(MEMBER)	Renames the member in the old name with the name of the member in the new name. The FILE portion of each name must be the same
RENAME logical-name FILE(MEMBER) LIBRARY/FILE(MEMBER)	Renames the member in the old name with the name of the member in the new name if the library specified in the old name is the same as the current library and both file name portions match
RENAME logical name LIBRARY/FILE(MEMBER) FILE(MEMBER)	Renames the member in the old name with the name of the member in the new name if the library specified in the new name is the same as the current library and both file name portions match

Table 18.5 rename Command Forms

CHAPTER 19

Remote Execution

19.1 Overview

This chapter provides information on Universal Data Mover (UDM) remote execution.

UDM provide two commands for remote execution:

- exec Command
- execsap Command

19.2 exec Command

19.2.1 Executing Remote Commands within UDM

If you have Universal Command (UCMD) Manager on the same system with the UDM Manager, you can execute system commands on remote machines using the **exec** command.

The **exec** command has the following format:

```
exec logical-name|host-name [cmd|cmdref|stc]=command [user=userid
     pwd=password] [port=port] [codepage=codepage] [file=filename]
     [xfile=filename] [key=key] [option=option] [mergelog=yes|no]
     [trace=yes|no] [input=data-element] [svropt=server-options]
     [stdout=data-element] [stderr=data-element]
```

The first parameter of the **exec** command is either:

- Logical name (**logical-name**) of a transfer server (valid only if a transfer session has been established)
- Host name (**host-name**) of the machine on which you want to execute the command.

Note: You must have the UCMD Server and Universal Broker installed on the machine on which the command is to be executed.

The second parameter is the command type, which is either:

- **cmd** (command)
- **cmdref** (command reference)
- **stc** (started task)

For any of these three types, the value (*command*) is the remote command to be executed. (See the Universal Command 4.2.0 Reference Guide for more information about command types.)

UDM must authenticate a user on the remote machine in order to execute a command.

- If a logical name is specified in the first parameter, the **user** and **pwd** values are inherited from the same options specified in the open command for that logical name. These inherited values can be overridden by specifying them explicitly in the **exec** command.
- If a host name is specified in the first parameter, the **user** and **pwd** values must be specified explicitly in the **exec** command.

The **port** and **codepage** values are inherited from the UDM Manager's configuration file unless overridden explicitly in the call to the **exec** command.

- **port** specifies which port the Universal Broker is listening on for the remote machine.
- **codepage** specifies to which codepage the output of the remote command is translated.

The **user**, **pwd**, **port**, and **codepage** parameters can be stored in an external file instead of being specified explicitly in the **exec** command.

- If a plain text file is used, use the **file** parameter to specify the name of this file.
- If the file was encrypted with Universal Encrypt, use the **xfile** parameter to specify the name of this file.

If an encryption key other than the Universal Encrypt default was used, specify that key with the **key** parameter.

These parameters, and the format of the file containing these parameters, work exactly like the corresponding option in the **open** command.

The **option** parameter is used to pass options to the UCMD Server (see the SCRIPT_OPTIONS option for UCMD Manager in the Universal Command 4.2.0 Reference Guide for more details).

Two streams of data come back from the **remote** command. By default, output from standard out and standard error of the remote command are written to standard out and standard error by the UDM Manager (SYSPRINT and SYSOUT, respectively, under z/OS). The **mergelog** option can be set to yes if you want both output streams written to the UDM transaction log (standard out under UNIX, Windows, and IBM i; SYSPRINT under z/OS).

By default, if the UDM Manager is invoked with tracing turned on, tracing will be turned on in the UCMD Manager when UDM invokes it via the **exec** command. Likewise, if **trace** is turned off in the UDM Manager, the UCMD Manager is invoked with tracing turned off. You can override this behavior for the UCMD Manager invocation by setting the **trace** option in the call to the **exec** command.

There are some commands that require input from standard input. To provide this input, you must create a data element with the data command containing the input. Specifying the name of the data element with the **input** parameter will cause the information in the data element to be sent over as standard input to the **remote** command.

The **svropt** parameter can be used to override UCMD Server options.

Note: UDM does not require a space before the server options, as does Universal Command.

The **stdout** and **stderr** parameters specify data elements to contain standard out and standard error, respectively, from the remote command. If the data elements do not exist, they are created. If the data elements do exist, they are overwritten with the output from the remote command. If the value portion refers to an existing non-data element variable or the name of a built-in variable (that is, any variable beginning with an underscore), an error is issued.

The **exec** command output will still be written to UDM stdout (the transaction log) and UDM stderr, where appropriate, even with the presence of the **stdout** and/or **stderr**.

19.2.2 Return Values

When the **exec** command is invoked, the return value from the **exec** command indicates whether or not UDM was able to invoke the remote command. The return value from the exec command will be 0 (none) if the remote command was invoked.

Upon successful invocation of the remote command, it might be useful to have the return value of the remote command itself in addition to whether or not the remote command could be executed. The remote command's return value is stored in the built-in variable **_execrc**.

19.2.3 exec Command Examples

The following example uses the **exec** command to execute a simple directory listing on a Windows machine that is part of a transfer session:

```
exec winmachine cmd="dir c:\"
```

The following example uses the **exec** command to invoke a started task:

```
exec mvsmachine stc="mytask,parm=$(TASK_PARM)"
```

The following example calls the **exec** command that uses a data element for input to the remote command:

```
# Define the data element

data shellinput
  echo "Comparing $(_file) with $(_file).old:"
  diff $(_file) $(_file).old
  exit
end

# Rename all existing files on the destination

forfiles dst=*
  rename dst $(_file) $(_file).old
end

# Copy the new files over and compare them

forfiles src=*
  copy src=$(_file)
  exec dst cmd="sh" input=shellinput
end
```

19.3 execsap Command

19.3.1 Triggering SAP Events within UDM

If you have a version of the Universal Connector (version 3.1.1 or later) on the same system with the UDM Manager, you can execute SAP events using the **execsap** command.

The **execsap** command has the following format:

```
execsap [host=host-name]|destination type=event|generic
        [eventid=event-id] [parm=event-parm] [client=client]
        [user=userid] [pwd=password] [codepage=codepage]
        [file=filename] [xfile=filename] [key=key] [mergelog=yes|no]
        [trace=yes|no]
```

Note: UDM does not support the **execsap** command for IBM i and Windows.

The first parameter of the **execsap** command is either:

- Host parameter with an SAP destination entry
- Name of a destination in your SAP RFC file

The **type** parameter specifies the type of action being performed. A specified type of **event** requires that an SAP event ID be specified with the **eventid** parameter.

Note: For version 4.2.0, the only valid type is **event**, which triggers an SAP event.

An event parameter can be passed to the SAP event using the **parm** parameter.

The **client** parameter specifies the SAP client.

UDM must authenticate a user SAP in order to execute an SAP action. The user ID and password can be specified with the **user** and **pwd** parameters, respectively.

The codepage is inherited from the UDM Manager's configuration file unless explicitly overridden in the call to the **execsap** command. The **codepage** specifies to which codepage the output of the remote command is translated.

The **user**, **pwd**, and **codepage** parameters can be stored in an external file instead of being specified explicitly in the **execsap** command syntax.

- If a plain text file is used, the **file** parameter specifies the name of this file.
- If the file was encrypted with Universal Encrypt, the **xfile** parameter specifies the name of this file.

If an encryption key was used other than Universal Encrypt's default, that key can be specified with the **key** parameter. This options and the format of the file containing the options work exactly like the corresponding option in the open command.

Two streams of data come back from the SAP execution. By default, output from standard out and standard error is written to standard out and standard error by the UDM Manager (SYSPRINT and SYSOUT, respectively, under z/OS). The **mergelog** option can be set to **yes** if you want both output streams written to the UDM transaction log (standard out under UNIX, Windows, and IBM i; SYSPRINT under z/OS).

By default, if the UDM Manager is invoked with tracing turned on, tracing will be turned on in Universal Connector (USAP) when UDM invokes it via the **execsap** command. Likewise, if trace is turned off in the UDM Manager, USAP is invoked with tracing turned off. You can override this behavior for the USAP invocation by setting the **trace** parameter in the call to the **execsap** command.

19.3.2 execsap Command Example

The following is an example of executing an SAP event using the **execsap** command:

```
execsap  sapdest type=event eventid=MYEVENT parm=MYPARM +
         user=me pwd=mypwd
```

CHAPTER 20

Return Code Processing

20.1 Overview

Universal Data Mover (UDM) return codes, particularly for batch operations, are used to gauge the degree of success of a job. Each job generates a return code that indicates the status of the job when it ended.

20.1.1 UDM Return Codes

Table 20.1, below, organizes UDM return codes into four categories.

Category	Value	Description
Success	0 / none	All commands completed successfully.
Warning	4 / warn	Non-critical error in operation.
Error	8 / error	UDM command failed to execute because it was: • Inappropriately issued (for example, a **copy** command was issued before a transfer session had been established). • Malformed; that is, grammatically correct but either: • Missing required parameters. • Containing invalid parameters.
Fatal	16 / fatal	Fatal error has occurred; UDM cannot continue and must exit. A fatal error is one that prevents UDM from running: • Failure to allocate memory. • Failure to initialize portions of the UDM application. • Parser errors (grammatically incorrect scripting language).

Table 20.1 UDM Return Codes

Each return code category has an integer value and a convenient value.

Processed commands return only integers as return code values. The convenient values can be used when setting return codes in the **`_rc`** and/or **`_halton`** variables via the **`set`** command (see Section 20.3.1 Return Codes in set (Set) Command).

20.2 Return Codes in UDM Built-In Variables

During processing, UDM keeps track of the return codes from processed commands.

_lastrc Variable

The **`_lastrc`** built-in variable holds the return code of the last command issued. It also has a special attribute, **`message`**, that contains a human-readable string indicating what happened with the last executed statement.

_rc Variable

The **`_rc`** built-in variable holds the highest-numbered return code that UDM has received (and placed in the **`_lastrc`** built-in variable) from all processed commands during the current session.

`_rc` also can be set via the **`set`** command (see Section 20.3 Setting Return Codes).

_halton Variable

The **`_halton`** built-in variable specifies a return code that, if equaled or exceeded by the return code in **`_rc`**, causes UDM to exit.

Otherwise, when UDM exits, it returns the highest-numbered return code that it received to the UDM Manager.

`_halton` only can be set via the **`set`** command (see Section 20.3 Setting Return Codes).

(For detailed information on these variable, see Chapter 15 UDM Scripting Language.)

20.3 Setting Return Codes

20.3.1 Return Codes in set (Set) Command

You can use the **set** command to manage UDM's return code and UDM's action based on this return code. The **set** command lets you set any of the following return code values (integer or convenient) in both the **_halton** variable and the **_rc** variable:

- 0 / none
- 4 / warn
- 8 / error
- 16 / fatal

The following example sets the value of **_rc** to 0 and the **_halton** condition to error:

```
set _rc=0 _halton=error
```

Stoneman's Tip

Issuing the set command by itself, with no parameters,
will display the values of all of the UDM Manager's internal variables
that can be set by the user.

The **set** command also can be used to set other UDM Manager variables. See Section 14.45 set for detailed information on using the **set** command.

Note: You cannot use the **set** command to set the **_lastrc** variable.

Issuing the set Command

1. If the **set** command is issued without any parameters (variables), all of the global variables and their current values are displayed.
2. If the **set** command is issued with variable names but no following equal signs (**=**), the values to which the variables resolve are displayed.
3. If the **set** command is issued with variable names followed by an equal signs (**=**) but no values, the values are set to an empty string.

20.3.2 Return Codes in return (Return) Command

You also can use the **return** command to set the return code value (integer only) in the **_rc** variable. (See Section 14.42 return for detailed information on using this command.)

CHAPTER 21

Additional Information

21.1 Overview

This chapter identifies additional information relative to Universal Data Mover (UDM).

Table 21.1 identifies this information and provides a link to its location in this chapter.

Information	Description	Page
Common File System Attributes	Attributes common to UDM on most operating systems.	462
z/OS Dynamic Allocation Attributes	z/OS dynamic allocation attributes that can be specified via the attrib (Attribute) command.	463
z/OS Attributes for Allocating Temporary Data Sets when Copying Load Modules	z/OS attributes that can be used for allocating temporary data sets when copying load modules via the copy command.	466
IBM i-Specific File System Attributes	Attributes that are unique to the LIB file system for IBM i.	467
Built-in Variables	All UDM built-in variables.	470
Global Variable Attributes	Attributes that can be used with the following types of UDM global variables: • All variables (user-defined or built-in) • Specific built-in variables • Logical Name built-in variables	473
Statements	Statements that can be used in the UDM scripting language.	475
`if` Statement Comparators	Comparators that can be used in an `if` statement to determine the type of comparison to be made between the left-hand and right-hand values.	476
Command Expression Operators	Operators for UDM command expressions.	477
Character Code Pages	Character code pages available for use with UDM.	479
SSL Cipher Suites	SSL Cipher Suites provided for use with UDM.	481

Table 21.1 Universal Data Mover - Additional Information

21.2 Common File System Attributes

Table 21.2 provides information on file system attributes that are common to UDM on most operating systems.

Attribute Name	Values	Description
createop	**append**, **new**, or **replace**	Determines how the file is to be created. If the value is **append**, the transferred data is appended to the data already in the destination file, if it exists. If the destination file does not exist, it will be created. If the value is **new**, the UDM will copy the file only if the destination file does not already exist. If the destination file does exist at the time the copy operation is initiated, the operation will return with an error. If the value is **replace**, UDM will overwrite the destination file it already exists. Otherwise UDM will create the file. The default value is **new**.
defext	Any sequence of characters valid for the destination file system.	If the source filename is being implicitly used as the destination filename, the sequence of characters specified by this attribute is appended to the end of the filename used to write the destination file. This occurs after the file extension has been truncated (if **truncext** is set to **yes**). By default, no default extension is defined. **Note:** The sequence of characters is appended verbatim. UDM does not add a dot character before the sequence, so if one is desired, it must be explicitly specified.
eol	Any sequence of valid text data	Specifies the end of line sequence used in text transfers. For the source side of a copy operation, excepting those from the z/OS **dd** and **dsn** file systems and the IBM i LIB file system, the end of line sequence is used to determine the end of each line of data. When the specified the sequence occurs in the data, UDM considers all data read up to that point (starting from the previous line) as a single line. Each line is transferred without the end of line sequence. On the destination side of a text transfer, the end of line sequence is appended to the end of each line before it is written. Two special character sequences can be used in any end of line sequence: • **\r** sequence indicates a carriage return. • **\n** sequence indicates a line feed. The default value of **eol** depends on the platform and the file system: • Under Windows, the default value is **\r\n**. • For UNIX platforms and the HFS file system under USS, the default value is \n. • Under z/OS for the **dd** and **dsn** file systems, and under IBM i for the LIB file system, the **eol** attribute is undefined. • Under IBM i for the **hfs** file system, the default is `FILE`, which makes end-of-line terminator consistent with file `ccsid`.

Attribute Name	Values	Description
linelen	A positive integer	Determines the maximum length of each line of data (record under the z/OS **dd** and **dsn** file systems) written. It applies only to the destination side of a transfer and is used in conjunction with the **lineop** and **padline** attributes. By default, **linelen** has a value of zero. Under Windows, UNIX, and the **hfs** file system, this means no line operation takes place. Under the z/OS **dd** and **dsn** file systems, the value **linelen** will be set equal to the logical record size used in allocating the destination file.
lineop	**none**, **stream**, **wrap**, or **trunc**	Sets the line operation for transferred lines (records under the z/OS **dd** and **dsn** file systems). For the line operation to be in play in the transfer, the value of **linelen** must not be zero (**linelen** is set automatically for the z/OS **dd** and **dsn** file systems to the logical record size if it is zero). If the value is **none**, each source line or record or data is written as it is received as a complete line or record. If the length of the source line is greater than that specified by **linelen**, UDM issues an error and the transfer operation is aborted. If the value is **stream**, the source data is treated as one long, single line of data. The source data is broken into multiple lines (records), each with a length of that specified by **linelen**. If the value is **trunc**, each source line longer than the value specified by **linelen** is truncated to be exactly **linelen** characters long. If the value is **wrap**, each source line longer than the value specified by **linelen** is broken up into multiple lines, each no longer than **linelen** characters long. Each segment is written out as a separate line (record). **Note:** If an end of line sequence is specified, the length of the sequence is not considered by UDM when determining the length of a line on the destination side. UDM only looks at the raw data that is transferred. By default the **lineop** attribute is not defined.
mode	Set of three numbers (0-7) or nothing	Note: This attribute is used only for UDM for UNIX. Specification for the mode (in UNIX parlance), or file permissions, of a file created by UDM in a copy operation. Existing files do not have their modes modified by UDM. They retain the file mode that they had before the copy operation was initiated. Each number in the set corresponds to one or more individuals for whom access is granted for the file: • First number Owner of the file. • Second number Users in the group assigned to the file. • Third number Everyone else. The value of each number is the sum of values representing file permissions: • 0 No permissions. • 1 Permission to execute the file. • 2 Permission to write to the file. • 4 Permission to read from the file. [By default, the mode attribute is not set. The default mode of a newly created file by UDM is dependent upon the user's umask or the mode of the source file in a UDM transfer.]

Attribute Name	Values	Description
padline	**none**, **null**, or **space**	Used in conjunction with **linelen**, when **linelen** is not zero. If the value is **none**, each line (record) of data written is not padded. If the value is **null**, each line of data is padded with null characters (hex value 0) at the end until the line is **linelen** characters in length. If the value is **space**, each line of data is padded with spaces at the end until the line is **linelen** characters in length. The default value is **none**.
trans	**yes** or **no**	Indicates whether or not a transactional file copy is to be performed. If the value is **yes**, the file is copied to a file with a temporary name that is renamed to the destination filename once the file has been successfully transferred. If the value is **no**, the file is copied directly to the specified destination filename. The default value is **no**. For z/OS and IBM i, **`trans`** is valid only under the **hfs** file system.
truncext	**yes** or **no**	Indicates whether or not the source filename's extension should be truncated if it is being used as the destination filename (no filename was explicitly specified on the destination side of the transfer operation). If the value of this attribute is **yes**, the extension is truncated. If the value is **no**, the filename is left untouched. The default value is **no**. **Note:** UDM considers a file extension to be the sequence of characters following the last dot (.) character in the filename. When an extension is truncated, the dot marks the beginning of the extension and is truncated as well. UDM will not consider the a dot character as the first character in a filename as indicating a file extension.
umask	A three-digit octal value	Valid only on the destination side of the transfer for the UNIX platform and the z/OS HFS file system (**hfs**). Specifies the file permissions mask used to create the destination file. Refer to the UNIX man page umask(1) for complete details. By default, the **umask** attribute is not defined. Note: In a UNIX-to-UNIX transfer, if the destination file does not exist, UDM will set the file permissions to match that of the source file once transfer has completed.
usefqn	**yes** or **no**	Valid only on the source side of the transfer for the z/OS platform. Specifies whether when copying a data set under the DSN file system, the fully qualified data set name is sent over as the source file name to be used by the destination if an explicit destination filename is not given. If set to no, only the part of the data set name matching the source mask in the copy operation is used as the destination filename. The default value is **no.**

Table 21.2 Common File System Attributes

For an explanation of how these attributes are used, see Chapter 16 UDM Transfer Operations.

21.3 z/OS Dynamic Allocation Attributes

Table 21.3, below, lists the z/OS dynamic allocation attributes that can be specified via the attrib (Attribute) command.

(For complete details on an allocation attribute, refer to the IBM JCL Reference.)

Attribute Name	Description
abnormaldisp	• Disposition of a data set after the job ends abnormally. • Equivalent to the third position sub-parameter of the JCL DISP parameter: DISP=(status,normaldisp,abnormaldisp). • Default is DELETE: it can be set with the UDM configuration option ALLOC_ABNORMAL_DISP.
avgrec	• Indication that the unit of allocation space specified with the **spaceunit** attribute is records and that the primary space and secondary space values are in units of 1's, K's, or M's. • Equivalent to the JCL AVGREC parameter: AVGREC=size. • No default or configuration option.
blksize	• Block size with which the data set is allocated. • Equivalent to the JCL BLKSIZE parameter: BLKSIZE=size. • Default is 27998: it can be set with the UDM configuration option ALLOC_BLKSIZE.
blkszlim	• Block size limit when there is not block size specified from any source. • Equivalent to the JCL BLKSZLIM parameter: BLKSZLIM=size. • No default or UDM configuration option.
dataclas	• SMS data class name. • Equivalent to the JCL DATACLAS parameter: DATACLAS=name. • No default: it can be set with the UDM configuration option ALLOC_DATACLAS.
datasetseq	• Data set sequence number that specifies the relative position of a tape data set on the volume. • Equivalent to the data set sequence sub-parameter of the JCL LABEL parameter: LABEL=(datasetseq,,,,). • No default or UDM configuration option.
ddndcbref	• DCB reference to a ddname. • Equivalent to the ddname sub-parameter of the JCL DCB parameter: DCB=ddname. • No default or UDM configuration option.
den	• Tape density to use. • Equivalent to the DEN sub-parameter of the JCL DCB parameter: DCB=DEN=density. • No default or UDM configuration option.
dirblocks	• Number of directory blocks to allocate for a partitioned data set. • Equivalent to the third positional parameter of the second positional parameter of the JCL SPACE parameter: SPACE=(, (,, dirblocks), . . .). • Default is 20: it can be set with the UDM configuration option ALLOC_DIR_BLOCKS.
dsndcbref	• DCB reference to a cataloged data set name. • Equivalent to the data set name sub-parameter of the JCL DCB parameter: DCB=dsn. • No default or UDM configuration option.

Attribute Name	Description
dsntype	• Type of SMS data set to allocate. • Equivalent to the JCL DSNTYPE parameter: DSNTYPE=type. • No default or UDM configuration option.
dsorg	• Data set organization with which the data set is allocated. • Equivalent to the JCL DSORG parameter: DSORG=org. • Default is PS: it can be set with the UDM configuration option ALLOC_DSORG.
expdt	• Expiration date of the data set. • Equivalent to the JCL EXPDT parameter: EXPDT=date. • No default or UDM configuration option.
label	• Data set label type used for mostly tape data sets. • Equivalent to the label sub-parameter of the JCL LABEL parameter: LABEL=(,label,,,). • No default or configuration option.
like	• SMS data set name from which to model data set attributes. • Equivalent to the JCL LIKE parameter: LIKE=dsname. • No default or UDM configuration option.
lrecl	• Logical record length with which the data set is allocated. • Equivalent to the JCL LRECL parameter: LRECL=len. • Default is 1024: it can be set with the UDM configuration option ALLOC_LRECL.
mgmtclas	• SMS management class name. • Equivalent to the JCL MGMTCLAS parameter: MGMTCLAS=name. • No default: it can be set with the UDM configuration option ALLOC_MGMTCLAS.
normaldisp	• Disposition of a data set after the job ends. • Equivalent to the JCL DISP parameter: DISP=(status,normaldisp,abnormaldisp). • Default is CATLG: it can be set with the UDM configuration option ALLOC_NORMAL_DISP.
password	• Password for password protected data sets. • No JCL equivalent.
primspace	• Primary amount of space to allocate for the data set. • Equivalent to the first positional parameter of the second positional parameter of the JCL SPACE parameter: SPACE=(, (primspace,), . . .). • Default is 15: it can be set with the UDM configuration option ALLOC_PRIM_SPACE.
pwdprotect	• Specification for whether or not the data set is password protected. • Equivalent to the PASSWORD or NOPWREAD sub-parameters of the JCL LABEL parameter: LABEL=(,,{PASSWORD|NOPWREAD},,). • Value must be either PASSWORD or NOPWREAD.
recfm	• Record format with which the data set is allocated. • Equivalent to the JCL RECFM parameter: RECFM=fmt. • Default is VB: it can be set with the UDM configuration option ALLOC_RECFM.
refdd	• ddname from which to copy SMS data set attributes. • Equivalent to the JCL REFDD parameter: REFDD=ddname. • No default or configuration option.
retpd	• Retention period of the data set. • Equivalent to the JCL RETPD parameter: RETPD=date. • No default or configuration option.

Attribute Name	Description
rlse	• Specification for whether or not to release unused space when the data set is unallocated. • Equivalent to the sub-parameter RLSE of the JCL SPACE parameter: SPACE=(,(,,),RLSE). • Default is no. There is no UDM configuration option. • Setting the attribute value to *yes* turns on the attribute.
secspace	• Secondary amount of space to allocate for the data set. • Equivalent to the second positional parameter of the second positional parameter of the JCL SPACE parameter: SPACE=(, (, secspace), . . .). • Default is 15: it can be set with the UDM configuration option ALLOC_SEC_SPACE.
spaceunit	• Allocation unit used to specify the space to allocate for the data set. • Equivalent to the first positional parameter of the JCL SPACE parameter: SPACE=(unit, . . .). • Default is TRK: it can be set with the UDM configuration option ALLOC_SPACE_UNIT.
status	• Status of the data set to be allocated. • Equivalent to the first positional parameter of the JCL DISP parameter: DISP=(status,normaldisp,abnormaldisp). • Default is: OLD for input and output data sets that exist, NEW for output data sets that don't exist. • Default input status can be set with UDM configuration option ALLOC_INPUT_STATUS. • Default output status can be set with UDM configuration option ALLOC_OUTPUT_STATUS.
storclas	• SMS storage class name. • Equivalent to the JCL STORCLAS parameter: STORCLAS=name. • No default: default value can be set with UDM configuration option ALLOC_STORCLAS.
unit	• Unit on which the data set is allocated. • Equivalent to the JCL UNIT parameter: UNIT=unit. • Default is SYSALLDA: it can be set with UDM configuration option ALLOC_UNIT.
unitcnt	• Number of units to allocate for a multi-volume data set. • Equivalent to the unit count sub-parameter JCL UNIT parameter: UNIT=(,unitcnt,). • No default or UDM configuration option.
volcnt	• Number of volumes to allocate for a multi-volume data set. • Equivalent to the volume count sub-parameter JCL VOL parameter: VOL=(,,,volcnt,). • No default or UDM configuration option.
volseq	• Volume sequence number on which a multi-volume data set starts. • Equivalent to the volume sequence number sub-parameter JCL VOL parameter: VOL=(,,volseq,,). • No default or UDM configuration option.
volser	• Volume serial number on which the data set is allocated. • Equivalent to the SER sub-parameter of the JCL VOL parameter: VOL=SER=volser. • No default: default value can be set with UDM configuration option ALLOC_VOLSER.

Table 21.3 z/OS attrib Command - Dynamic Allocation Attributes

21.4 z/OS Attributes for Allocating Temporary Data Sets when Copying Load Modules

Table 21.4, below, lists the z/OS attributes that can be used for allocating temporary data sets when copying load modules via the copy (Copy) command.

Attribute Name	Description
TMPVOLSER	Sets the volume on which temporary files will be allocated: • On the source side: specifies the location of the temporary sequential data set that will be transferred. • On the destination side: specifies the volume for the temporary transfer file as well as the volume used by the temporary staging PDS/E.
TMPPRIMSPACE	Specifies the amount of primary space used when allocating the temporary files. • On the source side: Affects the temporary sequential data set that will be transferred. • On the destination side: Used in allocating the temporary transfer file and the temporary staging PDS/E.
TMPSECSPACE	Specifies the amount of secondary space used when allocating the temporary files. • On the source side: Affects the temporary sequential data set that will be transferred. • On the destination side: Used in allocating the temporary transfer file and the temporary staging PDS/E.
TMPSPACEUNIT	Specifies the space unit used when allocating the temporary files. • On the source side: Affects the temporary sequential data set that will be transferred. • On the destination side: Used in allocating the temporary transfer file and the temporary staging PDS/E.
TMPDIRBLOCKS	Specifies the number of directory blocks used. • On the source side: n/a. • On the destination side: specifies the number of directory blocks used by the staging PDS/E.

Table 21.4 z/OS Attributes for Allocating Temporary Data Sets when Copying Load Modules

21.5 IBM i-Specific File System Attributes

21.5.1 LIB File System Attributes

Table 21.5 identifies attributes that are unique to the LIB file system for IBM i.

Name	Description	Source	Value	UDM Default	System Default	File Type
ACCPTH	Access path type		ARRIVAL, KEYED	NULL	ARRIVAL	SP
ALWDLT	Allow delete operation	√	YES, NO	NULL	YES	PF, SP
ALWUPD	Allow update operation	√	YES, NO	NULL	YES	PF, SP
ASPDEV	ASP device		ASP, ASPGRPPRI, SYSTEM, device name	NULL	ASP	LIB
ASPNUM	ASP number		LIBASP, 1-32, ASPDEV	NULL	LIBASP for SAVF, 1 for LIB	LIB, SAVF
AUT	Authority		LIBCRTAUT, ALL, CHANGE, EXCLUDE, USE [2]	NULL	LIBCRTAUT for LIB,PF,SP EXCLUDE for SAVF	LIB, PF, SP, SAVF
CCSID *	CCSID of the file (source physical files only). For data physical files, the DDS (if one is given) determines its CCSID; if no DDS is given, the value is 65535.	√	EBCDIC CCSIDs	CODEPAGE		SP
CRTAUT	Create authority		SYSVAL, ALL, CHANGE, EXCLUDE, USE, authority name	NULL		LIB
DDSLIB	Library of the DDS used to describe the file	√		empty string		PF
DDSFILE	File of the DDS used to describe the file	√		empty string		PF
DDSMBR	Member of the DDS used to describe the file	√		empty string		PF
DLTPCT	Maximum percentage of deleted records allowed		1-100, NONE	NULL	NONE	PF
EXPDATE	Expiration date for member	√	date, NONE	NULL	NONE	PF, SP
FILETYPE	Type of file to create when creating a new file	√	DATA, SRC, SAVF	DATA		PF, SP, SAVF

Name	Description	Source	Value	UDM Default	System Default	File Type
FRCRATIO	Records to a force write		integer, NONE	NULL	NONE	PF, SP
GENLVL	Generation severity level		0-30	NULL	20	PF
LIBTYPE	Type of library created when creating a library	√	PROD, TEST	PROD		LIB
LVLCHK	Record format level check	√	YES, NO	NULL	YES	PF
MAXMBRS	Maximum number of members	√	integer, NOMAX	NULL	1 for PF, NOMAX for SP	PF, SP
MAXRCDS	Maximum number of records		1-2146762800, NOMAX	NULL		SAVF
OPTION	Source listing options		SRC, NOSRC, SOURCE, NOSOURCE, LIST, NOLIST, SECLVL, NOSECLVL, EVENTF, NOEVENTF (up to four repetitions)	empty string		PF
RCDLEN	Record length if no DDS is used	√	integer	92		PF, SP
REUSEDLT	Reuse deleted records	√	YES, NO	NULL	NO	PF
SEQSTART	Beginning sequence number used when writing to a source physical file	√	0000.01 – 9999.99	1.00		SP
SEQINCR	Amount to increment sequence number by when writing a record to a source physical file	√	00.01 – 99..99	1.00		SP
SHARE	Share open data path	√	YES, NO	NULL	NO	PF, SP, SAVF
SIZE	Member size	√	• Single values: NOMAX • Other values: Comma-separa ted element list • Element 1: Initial number of records 1-2147483646, • Element 2: Increment number of records Integer, • Element 3: Maximum increments Integer (EX: 10000,1000,3)	size_attrib configuration file entry if provided; otherwise empty string	10000,1000, 3 for PF, 10000,1000, 499 for SP	PF, SP

Name	Description	Source	Value	UDM Default	System Default	File Type
USESRCSEQ	Sequence number and modification date information: • On Source side: retain this information when copying a source physical file • On Destination side: Record data includes this information	√	YES, NO	NO		SP
WAITFILE	Maximum file wait time		integer, IMMED, CLS	NULL	30 for PF IMMED for SP, SAVF	PF, SP, SAVF
WAITRCD	Maximum record wait time		integer, IMMED, NOMAX	NULL	60	PF, SP
* With CCSID set to CODEPAGE, when the UDM CCSID attribute is not set either explicitly or implicitly via an IBM i to IBM i file transfer, the CCSID associated with the code page via the code page to CCSID mapping tables gets used as the CCSID attribute value. One implication is that, by default, files may be created with the CCSID associated with the codepage option.						

Table 21.5 IBM i-Specific LIB File Attributes for Creating New Files

21.5.2 HFS Attributes

Table 21.6 identifies attributes that are unique to the HFS file system for IBM i.

(Currently, there is only one HFS unique attribute, CCSID.)

Name	Description	Source	Value	UDM Default	System Default	File Type
CCSID	CCSID of the file	√	EBCDIC and ASCII CCSIDs	CODEPAGE		stream

Table 21.6 IBM i-Specific HFS File Attributes for Creating New Files

21.6 Built-In Variables

Table 21.7 lists all of the UDM built-in variables.

Variable	Description
_date	Displays the current date in the format appropriate for the system's locale.
_echo	Specification for whether or not a command is echoed prior to processing.
_execrc	Holds the value of the process executed by the last exec command issued.
_file	Name of the file for the current iteration in a **`forfiles`** loop.
_halton	Return code value that causes UDM to terminate if it is greater than 0 and is equalled or exceeded by the return code value in the _rc variable.
_keepalive	Interval at which keep-alive messages are sent form the UDM Manager to transfer servers.
_lastmsg	Contains all of the messages written in the transaction log for the last network- or file-oriented command issued.
_lastrc	Holds the return code of the last command issued and, optionally, an indication of what happened with the last executed statement.
_lines	Specification for whether or not the line number is printed with the error if a command cannot be parsed or is malformed.
_path	Absolute path of the file for the current iteration in a **`forfiles`** loop.
_rc	Current UDM return code.
_time	Current time.
_uuid	Generates a UUID.

Table 21.7 Built-In Variables

For an explanation of how these variables are used, see Section 15.6 UDM Variables.

21.7 _file Built-in Variable – Special Attributes

Table 21.8 lists all of the special attributes for the _file built-in variable.

Attribute Name	Description
accessdate	Date on which the file was last accessed. Format (ISO 8601) is yyyy-mm-dd.
accesstime	Time when the file was last accessed. Format (ISO 860) is hh:mm:ss.
accesstimestamp	Combination of **accessdate** and **accesstime**: yyyy-mm-dd hh:mm:ss. If the file does not have an access time, but does have a access date, 00:00:00 is used for the time portion.
createdate	Date on which the file was created. Format (ISO 8601) format of yyyy-mm-dd.
createtime	Time when the file was created. Format (ISO 8601) is hh:mm:ss.
createtimestamp	Combination of **createdate** and **createtime**: yyyy-mm-dd hh:mm:ss. If the file does not have a creation time, but does have a creation date, 00:00:00 is used for the time portion.
moddate	Date on which the file was last modified (referenced for z/OS). Format (ISO 8601) is yyyy-mm-dd.
modtime	Time when the file was last modified (referenced for z/OS). Format (ISO 8601) is hh:mm:ss.
modtimestamp	Combination of **moddate** and **modtime**: yyyy-mm-dd hh:mm:ss. If the file does not have a modification time, but does have a modification date, 00:00:00 is used for the time portion.
name	Name of the file (same as referencing **_file** itself without any attributes).
size	Size of the file (in bytes).
type	Type of file. Values are: • file • directory (also used for PDSs under z/OS) • unknown **type** has meaning in a **forfiles** statement under IBM i in the LIB file system: • If the value of _file.type is directory, the file type is a Physical file. • If the value of _file.type is file, the file type is a Save file.

Table 21.8 _file Built-in Variable – Special Attributes

21.8 Global Variable Attributes

Table 21.9 lists the attributes that can be used with the following types of UDM global variables:

- All variables (user-defined or built-in)
- Specific built-in variables
- Logical Name built-in variables

Variable	Attribute	Attribute Description
(all)	exists	Expands to **yes** if a variable with that name exists at any scope; it expands to **no** if no variable with that name exists.
(all)	length	Expands to the length of the variable's value.
_date	day	Resolves to the day of the week.
_date	month	Resolves to the current month.
_date	dd	Resolves to a two-digit day of the month.
_date	ddd	Resolves to the Julian day.
_date	mm	Resolves to the two-digit month of the year.
_date	yy	Prints the two-digit year.
_date	ww	Resolves to the two-digit current week of the year. (The value of ww is zero-based, not one-based. That is, the first week of the year is 0, the second week is 1, the third week is 2, and so on.)
_date	yyyy	Resolves to the four-digit year.
_file	type	Type of file contained in the **`file`** variable: file, directory (also used for PDSs under z/OS), or unknown. **`type`** also has meaning in a **`forfiles`** statement under IBM i in the LIB file system: • If the value of **`_file.type`** is **`directory`**, the file type is a Physical file. • If the value of **`_file.type`** is **`file`**, the file type is a Save file.
_lastrc	message	Human-readable string indicating what happened with the last executed statement. • If a command could not be executed or had improper values, the value of **`_lastrc.message`** is ERROR. • If a command successfully executed, the value of **`_lastrc.message`** is either SUCCESS or some other message (depending upon the command).
_lastrc	result	Integer value that indicates the result of the last command executed. The meaning of this value depends on the command. Unless otherwise stated: • -1 indicates failure. • 0 or a positive value indicates success.
_time	hh	Resolves to the two-digit hour (24-hour time).
_time	mm	Resolves to the two-digit minute.
_time	ss	Number of seconds that have elapsed since the current minute.

Variable	Attribute	Attribute Description
_time	hs	Resolves to the number of hundredths of a second that have elapsed since the last second.
(logical name)	host	Contains the host name of the transfer server.
(logical name)	port	Holds the port used to connect to the transfer server over.
(logical name)	user	Contains the userid used to sign into the transfer server.

Table 21.9 UDM Global Variable Attributes

For an explanation of how these attributes are used in the variables, see Section 15.6 UDM Variables in Chapter 15 UDM Scripting Language.

21.9 UDM Statements

Table 21.10 lists all of the statements that can be used in the Universal Data Mover scripting language.

Statement	Description
if	Adds conditional branching of UDM commands. An `if` statement consists of: 1. Comparison operation. 2. Series of UDM commands that are carried out if the comparison operation evaluates to true. 3. **end** statement that indicates the end of the `if` statement. A comparison consists of three parts: 1. Left-hand value 2. Comparator (see Section 21.10 if Statement Comparators) 3. Right-hand value The left-hand and right-hand values can be either: • Variable reference • Variable attribute • Constant The syntax is: **`if comparison`** … **`UDM commands`** … **end** If the comparison does not evaluate to true, UDM picks up execution from the line after the **end** statement.
else	Provides an alternate path, when used as part of an `if` statement, if the comparison evaluates to false. The syntax is: **`if expression`** … **`[else`** **…]** **end** In this `if` statement, the parameter for `if` is an expression. If the expression evaluates to a value that is not equal to zero, the positive branch is taken; otherwise, the negative (`else`) branch is taken.

Statement	Description
while	Implements a simple **`while`** loop. The syntax is: **`while expression`** `...` **`end`** In this case, the loop iterates (executing the commands between the while and end statements) as long as the expression evaluates to a value that is not zero. If the expression evaluates to a value of zero, code execution picks up at the point immediately following the end of the **`while`** loop.

fordata	Iterates through a data element, once for each line.
	For each iteration, a variable provided by the user is set to hold the contents of the line in the data element corresponding to the current iteration.
	The syntax is:
	`fordata variable-name=data-element`
	`...`
	`end`

forfiles	Iterates through a series of statements for each file found that matches a given file specification.
	The syntax is:
	`forfiles logical_name=file_spec`
	`         [sortby=attribute-name[,ascending|descending]]`
	`    ...`
	`    UDM commands`
	`    ...`
	`end`

subroutine	Names a subroutine and defines the script code that becomes associated with that subroutine name.
	The syntax is:
	`subroutine name`
	`[script line 1}`
	`...`
	`{script line 2]`
	`endsub`

callsub	Carries out the work of lines of script associated with a subroutine. The syntax is: **`callsub name`**

Table 21.10 UDM Statements

For an explanation of how these statements are used, see Section 15 UDM Scripting Language.

21.10 if Statement Comparators

Table 21.11 lists all of the comparators that can be used in an `if` statement to determine the type of comparison to be made between the left-hand and right-hand values.

Comparator	Description
EQ (Equal)	Evaluates to true if the left-hand and right-hand values are equal. If one or more of the values contains alpha characters (non-numeric), the comparison is case insensitive.
NE (Not Equal)	Evaluates to true if the left-hand value is not equal to the right-hand value. If one or more of the values contains alpha characters (non-numeric), the comparison is case insensitive.
LT (Less Than)	Evaluates to true if the left-hand value is less than the right-hand value. LT performs a numeric comparison.
GT (Greater Than)	Evaluates to true if the left-hand value is greater than the right-hand value. GT performs a numeric comparison.
LE (Less Than or Equal)	Evaluates to true if the left-hand value is less than or equal to the right-hand value. LE performs a numeric comparison.
GE (Greater Than or Equal)	Evaluates to true if the left-hand value is greater than or equal to the right-hand value. GE performs a numeric comparison.

Table 21.11 if Statement Comparators

For an explanation of how these comparators are used, see Section 15.7.1 Comparison Operations in Section 15.7 if Statement.

21.11 UDM Command Expression Operators

Table 21.12 identifies and describes all of the operators for UDM command expressions.

Operator	Description
EQ	Compares the value on the left to the value on the right. If the two are equal, the result is 1, otherwise it is 0. Both the left an right values can be strings or numbers. If they are strings, the comparison is case insensitive.
NE	Works like the equal operator, except that it results in 1 if the left and right value are not equal and 0 if they are.
LT	Results in a value of 1 if the left value is less than the right value, otherwise it results in 0. This is a numeric operator.
GT	Results in a value of 1 if the left value is greater than the right value, otherwise it results in 0. This is a numeric operator.
LE	Results in a value of 1 if the left value is less than or equal to the right value, otherwise it results in 0. This is a numeric operator.
GE	Results in a value of 1 if the left value is greater than or equal to the right value, otherwise it results in 0. This is a numeric operator.
AND	Results in a value of 1 if both the left value and right value are not 0, otherwise it results in 0.
OR	Results in a value of 1 if either the left value or the right value are not 0, otherwise it results in 0.
XOR	Results in a value of 1 if either the left or the right values are not 0, but not both. If both the left and right values are 0, then the result of the XOR operator is 0.
NOT	Unlike all of the operators, the NOT operator has only one operand that appears to the right of the NOT operator. This operation evaluates to one if the operand is zero and zero if the operand is non-zero.
+	Result is the sum of the left and right values.
-	Result is subtracting the right value from the left value.
*	Result is the product of the left and right values.
/	Result is the left value divided by the right value. Assuming integer-only math, the remainder is discarded.
%	Result is the remainder of the left value divided by the right value.

Table 21.12 UDM Command Expressions - Operators

21.12 Character Code Pages

Table 21.13 identifies the character code pages provided by Stonebranch Inc. for use with Stonebranch Solutions on each supported operating system.

Code Page	CCSID	z/OS	UNIX	Windows	IBM i		HP NonStop
					HFS	LIB	
IBM037	037	√			√	√	
IBM273	273	√			√	√	
IBM277	277	√			√	√	
IBM278	278	√			√	√	
IBM280	280	√			√	√	
IBM284	284	√			√	√	
IBM500	500	√			√	√	
IBM875	875	√					
IBM1047							
IBM1140	1140	√			√	√	
IBM1141	1141	√			√	√	
IBM1142	1142	√			√	√	
IBM1143	1143	√			√	√	
IBM1144	1144	√			√	√	
IBM1145	1145	√			√	√	
IBM1146	1146	√			√	√	
IBM1147	1147	√			√	√	
IBM1148	1148	√			√	√	
IBM4971	4971	√					
ISO8859-1	819		√	√	√		√
ISO8859-2	912		√	√	√		√
ISO8859-3	913		√	√	√		√
ISO8859-4	914		√	√	√		√
ISO8859-5	915		√	√	√		√
ISO8859-6	1089		√	√	√		√
ISO8859-7	813		√	√	√		√
ISO8859-8	916		√	√	√		√
ISO8859-9	920		√	√	√		√
ISO8859-10			√	√	√		√
ISO8859-13	921		√	√	√		√
ISO8859-14			√	√	√		√
ISO8859-15	923		√	√	√		√
PC437	437			√	√		

Code Page	CCSID	z/OS	UNIX	Windows	IBM i		HP NonStop
					HFS	LIB	
PC737	737			√	√		
PC775	775			√	√		
PC850	850			√	√		
PC852	852			√	√		
PC855	855			√	√		
PC857	857			√	√		
PC860	860			√	√		
PC861	861			√	√		
PC862	862			√	√		
PC863	863			√	√		
PC864	864			√	√		
PC865	865			√	√		
PC866	866			√	√		
PC869	869			√	√		
PC874	874			√	√		
WIN1250	1250			√	√		
WIN1251	1251			√	√		
WIN1252	1252			√	√		
WIN1253	1253			√	√		
WIN1254	1254			√	√		
WIN1255	1255			√	√		
WIN1256	1256			√	√		
WIN1257	1257			√	√		
WIN1258	1258			√	√		

Table 21.13 Character Code Pages

21.13 UTT Files

Table 21.14 identifies the Universal Translate Table (UTT) files that are used to translate between Unicode and the local single-byte code page.

Operating System	UTT File Location
IBM i	UTT files are located in the source physical file **`UNIVERSAL/UNVNLS`**. *codepage* is the member name of the UTT file.
z/OS	UTT files are located in the library allocated to the **`UNVNLS`** ddname. *codepage* is the member name of the UTT file.
UNIX	UTT files are located in the **`nls`** subdirectory of the installation directory. *codepage* is the base file name of the UTT file. All UTT files end with an extension of **`.utt`**.
Windows	UTT files are located in the NLS subdirectory of the installation directory. *codepage* is the base file name of the UTT file. All UTT files end with an extension of **`.utt`**.

Table 21.14 UTT File Locations

21.14 SSL Cipher Suites

Table 21.15 identifies all of SSL cipher suites provided by Stonebranch Inc. for use with UDM.

Cipher Suite	Description
RC4-SHA	128-bit RC4 encryption and SHA-1 message digest
RC4-MD5	128-bit RC4 encryption and MD5 message digest
AES256-SHA	256-bit AES encryption and SHA-1 message digest
AES128-SHA	128-bit AES encryption and SHA-1 message digest
DES-CBC3-SHA	128-bit Triple-DES encryption and SHA-1 message digest
DES-CBC-SHA	128-bit DES encryption and SHA-1 message digest
NULL-SHA	No encryption and SHA-1 message digest
NULL-MD5	No encryption and MD5 message digest
NULL-NULL *	UDM pseudo cipher used in conjunction with the open command encrypt option to disable SSL.
* UDM Manager ignores the NULL-NULL pseudo cipher.	

Table 21.15 SSL Cipher Suites for UDM

APPENDIX A

Customer Support

Stonebranch, Inc. provides customer support, via telephone and e-mail, for Universal Data Mover and all Stonebranch Solutions 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

950 North Point Parkway, Suite 200

Alpharetta, Georgia 30005

U.S.A.