

Content Server Administrator's Guide



Version 5.3 SP1
September 2005

Copyright © 1994-2005 EMC Corporation

Table of Contents

Preface	25
Chapter 1 Introduction	27
Essential concepts.....	28
What comprises an installation?	28
Configuration choices	28
Configuration objects.....	29
Administration tasks	30
User privilege requirements for administration tasks	32
Administration interfaces.....	32
Starting Documentum Administrator.....	32
Using the Content Server Manager on Windows.....	33
Using the API for administrative tasks	33
Using DQL for administrative tasks	33
Documentum tool suite.....	34
Administration methods	34
The dm_error utility	34
Viewing connected users.....	35
Where to look for more information.....	35
Chapter 2 Content Repositories	37
Essential concepts.....	38
How are repositories and servers connected?	38
Repository configuration.....	38
Adding additional repositories	39
Procedure	40
Configuring the new repository for use with Media Transformation Services	40
Contents of new repositories	41
Managing cabinets and folders	43
Public and private cabinets.....	43
Home cabinets.....	44
Creating folders and cabinets	44
Changing and deleting folders and cabinets	44
Setting the dd_locales attribute.....	45
Manipulating type indexes.....	45
Alternate locations for object-type tables on Oracle and DB2	46
Dumping and loading a repository	46
Code page compatibility issues.....	47
Supporting object types.....	47
Dumping a repository.....	48
Dumping objects under retention	48
Dumping an entire repository	49

Dumping specific objects.....	49
Setting the type attribute.....	49
Setting the predicate attributes.....	51
Content files and dumping.....	52
Dumping without content.....	52
Including content.....	53
Content compression.....	53
Setting the cache size.....	53
Using non-restartable dump.....	54
Using a script to create a dump file.....	54
Sample script for a full repository dump with content included.....	55
Sample script for a partial repository dump.....	55
If the server crashes during a dump operation.....	57
Moving the dump file.....	57
Loading a repository.....	57
Refreshing repository objects from a dump file.....	58
Loading job objects.....	58
Loading registered tables.....	59
Turning off save event generation during load operations.....	59
Loading a new repository.....	59
The preLoad utility.....	60
The load procedure for new repositories.....	60
DocApps.....	62
Generating dump and load trace messages.....	62
Creating location and mount point objects.....	62
Location objects.....	63
Mount point objects.....	65
Platform aliases.....	65
Format objects.....	66
The topic_transform and topic_format attributes.....	66
The DOS_extension attribute.....	67
The format_class attribute.....	67
Listing current format objects.....	67
Adding format objects.....	68
Using DQL.....	68
Using the API.....	68
Rich media formats.....	69
Modifying formats.....	69
Using DQL.....	70
Using the API.....	70
Deleting formats.....	71
Using DQL.....	71
Using the API.....	71
Alias sets.....	72
Creating an alias set.....	72
Modifying or deleting an alias set.....	72
Working with object types.....	73
Creating a user-defined type.....	73
Using DQL.....	73
Modifying an object type.....	74
Note on deleting attributes.....	74
Using DQL.....	75
Changing the default permissions or default storage area.....	75
Adding an attribute.....	75
Deleting an attribute.....	76
Lengthening a string attribute.....	76

Deleting a type	76
Cleaning up repositories	77
Maintaining query performance	80
Configuring repository-level package name control	80
Configuring a repository printer on Windows	81
Chapter 3 Servers	83
Overview of servers	83
Server threads (Windows)	84
Parent servers and session servers (UNIX)	84
Configuration	84
Multiple servers	85
Servers, connection brokers, and clients	85
The agent exec process	85
ACS servers	86
Internationalization	86
The dm_start_docbase script (UNIX)	87
The server.ini file	88
SERVER_STARTUP section	89
DOCBROKER_PROJECTION_TARGET sections	91
FUNCTION_SPECIFIC_STORAGE and TYPE_SPECIFIC_STORAGE sections	92
FUNCTION_EXTENT_SIZE and TYPE_EXTENT_SIZE sections	93
Keys you provide at installation	94
docbase_id	94
docbase_name	94
database_owner	94
database_conn	95
Oracle database_conn value	95
Sybase database_conn value	95
MS SQLServer database_conn value	95
DB2 database_conn value	95
database_name	96
Sybase database_name value	96
MS SQL Server database_name value	96
service	96
host	96
Optional keys	96
acl_update_threshold	97
check_user_interval	97
commit_read_operations	97
data_store and index_store	98
enable_workitem_mgmt	98
gethostbyaddr	99
history_sessions and history_cutoff	99
max_ftacl_cache_size	99
max_sessions_heap_size	99
saveasnew_retain_source_group	100
umask (UNIX only)	100
upd_last_chg_time_from_db	100
use_group_address	100
validate_database_user	101
The default keys	101
client_session_timeout	101
concurrent_sessions	101
database_refresh_interval	102

distinct_query_results	102
enforce_four_digit_year	102
ignore_client_domain (Windows only).....	103
mail_notification	103
method_server_enabled	103
method_server_threads.....	103
preserve_existing_types	104
rdbms_connect_retry_timeout.....	104
server_config_name.....	104
server_startup_sleep_time.....	104
start_index_agents.....	104
ticket_multiplier.....	105
update_access_date	105
user_auth_case.....	105
user_auth_target (Windows only).....	105
use_estimate_search	106
wait_for_connect_timeout.....	106
Moving the server executable (UNIX only).....	106
Changing default operating system permits on directories and files (UNIX only)	106
Changing a server's configuration.....	107
Modifying the server.ini file	107
Modifying the server config object	108
Setting the secure connection mode	109
Restarting a server	109
Starting additional servers.....	110
Configuration requirements	111
Creating a shut-down script (UNIX only).....	111
Communicating with connection brokers	112
Defining connection broker projection targets.....	112
Definitions in the server config object.....	113
Definitions in the server.ini file	114
Setting the checkpoint interval.....	114
Using the API.....	115
Setting the keep entry interval	116
Using the API.....	116
Defining server proximity	117
Shutting down a server	118
Using the dm_shutdown_repository script (UNIX only)	118
Using the Documentum Content Server Manager (Windows only).....	119
Using the Windows user interface (Windows only).....	119
Using the Shutdown method.....	119
Stopping a session server	120
Server log files.....	121
Server load balancing and failover	122
Clearing the server common area.....	123
Adding additional servlets to the Java method server.....	123
Configuring the workflow agent.....	124
Changing the number of worker sessions	124
Changing the sleep interval.....	124
Disabling the workflow agent.....	125
Tracing the workflow agent.....	125

	Recovering automatic activity work items on Content Server failure	125
Chapter 4	Methods and Jobs	127
	Introducing methods	127
	Execution agents	128
	dmbasic method server	128
	Java method server	128
	Content Server	129
	Choosing the execution agent.....	129
	Performance considerations.....	130
	Security considerations	130
	Tracing options for methods.....	131
	Defining the java.ini file (UNIX only)	132
	java_library_path.....	132
	java_version	132
	java_classpath	132
	java_alias_file	133
	java_disabled	133
	Enabling the dmbasic method server	133
	Configuring the worker threads in the dmbasic method server	133
	Implementing a method.....	134
	Creating a method to be executed by Content Server or the dmbasic method server	134
	General guideline for the script or program.....	134
	Script or program return values for workflow methods.....	135
	Calling Java or DFC in a Docbasic script.....	135
	User account (UNIX only)	135
	dmbasic executables (UNIX only)	135
	Locating the java.ini file (UNIX only)	135
	Sample code.....	136
	Recording the output.....	137
	Setting method object attributes for Content Server execution.....	138
	Setting method object attributes for dmbasic method server execution	138
	Creating a method to be executed by the Java method server.....	138
	General guideline for the script or program.....	139
	Script or program return values for workflow methods.....	139
	Recording the output.....	139
	Storing Java methods	140
	Setting method object attributes for Java method server execution	140
	Additional guidelines for the Java method server	140
	Creating a method object.....	141
	Using DQL.....	141
	Defining success return codes and success status	142
	Executing a method on demand	142
	Creating jobs and job sequences.....	143
	Introducing jobs	143
	Introducing job sequences.....	143
	Repository implementation	144
	Job sequence execution	144
	Determining success for invoked jobs	145
	The repository connection file.....	145
	The agent exec process.....	146
	Creating a job	146

Creating jobs in a multi-server environment	147
Using DQL to create a job.....	147
Creating a job sequence.....	147
Scheduling jobs	148
Defining a job schedule	148
Passing arguments.....	149
The run_now attribute	150
Managing jobs.....	151
Activating or inactivating a job	151
Disabling all jobs	151
Modifying agent exec behavior.....	152
Setting the polling interval	152
Setting the number of jobs in a polling cycle	152
Turning on tracing for the agent exec process	153
Creating and maintaining a repository connection file for job sequences.....	153
Specifying the server connect string	153
Commas and backslashes in the entries	154
The dcf_edit utility	154
Recovering from a job sequence failure	156
Interpreting a job sequence status report	156
Executing dm_run_dependent_jobs independently.....	157
Chapter 5 Clients and Sessions	159
Essential concepts.....	159
What is a client?	160
Client configuration.....	160
The dmcl.ini file.....	160
What is a session?	161
Session configuration.....	162
Session config object	162
Api config object.....	162
Server config object.....	162
Connection config object	163
Session connection attempts and timeouts.....	163
Session subconnections	163
Viewing files	164
Defining connection brokers for connection requests.....	164
Format of dmcl.ini sections.....	165
Failover and load balancing.....	166
Bypassing the connection broker	166
Requesting a specific server connection.....	167
Requesting a server by name	167
Requesting a server on a specific host.....	167
Requesting a server by name on a specific host	168
Requesting a native or secure connection	168
Limiting which clients can access a repository	169
Connection pooling	170
Enabling connection pooling	170
Setting the session recycle limit	170
Removing free connections from the pPool.....	171
Configuring login tickets.....	171
Setting ticket validity period.....	171
Setting the ticket cache size for Content Server	171
Changing a session's configuration	172

General procedure	172
The scope of changes	172
The Restart and Reinit methods	173
The dmcl.ini file	174
Adding keys to the file	174
Adding comments to the file	174
The DMCL_MASTER section	174
The DMAPI_CONFIGURATION section	174
The DOCBROKER sections	180
The DOCBASE_OVERRIDE_ <i>docbase</i> section	180
Finding the dmcl.ini file	181
Content Server installations	181
Desktop	181
WDK applications	181
Changing the dmcl.ini file	181
Changing the assigned default operating system permissions (UNIX only)	182
Defining short date formats	182
Disabling or enabling the client local area	183
Scope of the change	185
Disabling local area use	185
In the current repository session	185
For all repository sessions	186
Enabling local area use	187
In the current repository session	187
In all future repository sessions	188
Changing the local area directory	188
In the current API session	188
For the current repository session	188
To affect future repository sessions	189
To Affect Current and Future Repository Sessions	189
In all future repository sessions	190
Monitoring local area disk space	190
Removing content from local areas	191
Manual clean up	191
Managing persistent client caches	192
Enabling and disabling persistent client caching	192
For a repository	192
For client sessions	193
Creating cache config objects	194
Defining the cached data set	194
Defining the server check interval	195
Defining the client check interval	196
Manually forcing refreshes	196
Flushing a persistent cache	196
Setting the client_pcaching_change attribute	197
Automating cache config data validation	197
Overriding consistency checking rules	198
Defining the persistent cache write interval	198
Troubleshooting persistent caching	199
Using persistent client caching with a pre-5.2 Content Server	200
Configuring DMCL exception handling	201
Configuring the default behavior	201
Enabling the DMCL to continue on exception	201
Generating DMCL exception trace files	202

Chapter 6	Connection Brokers	205
	An overview of connection brokers.....	205
	How many connection brokers are there?.....	206
	What information does a connection broker have?.....	206
	How does a connection broker get information?.....	207
	How do I locate connection brokers?.....	207
	Connection broker configuration options	207
	Servers and connection brokers	208
	Clients and connection brokers.....	208
	Failover for connection brokers	209
	Load balancing for connection brokers.....	209
	Configuring a connection broker	209
	Connection broker initialization file	210
	Invoking the initialization file	211
	Configuring shutdown security (UNIX only).....	211
	Restricting server access.....	211
	Translating IP addresses.....	212
	Restarting a connection broker	213
	Windows platforms	213
	UNIX platforms.....	214
	Starting additional connection brokers.....	214
	Shutting down a connection broker	215
	Windows platforms	216
	UNIX platforms.....	216
	Multiple connection brokers on UNIX.....	217
	Obtaining information from a connection broker	218
	The Getservermap method	218
	The Getdocbasemap method	218
	Obtaining Information about Connection Brokers.....	219
	Using the Getdocbrokermap method	219
	Using the Get method	220
	Deleting server information.....	220
Chapter 7	Content Management	221
	Storage area options	221
	File store storage areas	223
	Public and private file store areas	223
	Content encryption.....	224
	Content compression	224
	Content duplication checking and prevention.....	224
	How checking and prevention work	225
	Supporting attributes.....	225
	content_hash_mode and content_dupl_pref	225
	r_content_hash	226
	Tracing duplication checking and prevention	227
	Digital shredding	227
	Content-addressed storage.....	227
	Configuration options.....	228
	Retention periods	228
	Compression.....	229
	Whether to link or embed the content in the C-clip	229
	Configuring write retries.....	230
	Blob store storage areas.....	230
	Turbo storage	231

Distributed storage areas.....	231
External storage areas	232
Use constraints	232
Types of external storage areas	232
Plug-in objects for external storage	233
Linked store storage areas.....	233
Summary of storage area configuration options	234
Content and full-text indexes.....	235
How objects, contents, and storage are connected	236
Allocating content to storage areas	237
Using content assignment policies	237
What content assignment policies are.....	237
Creating content assignment policies	238
Enforcement of assignment policies	238
DFC assignment policy information cache.....	239
Internal implementation of assignment policies	239
Algorithm used by the DFC policy engine	240
Assignment policy administration	241
Using the default storage algorithm	241
Primary content algorithm	241
Rendition algorithm.....	243
Content retention	243
System-defined storage areas	244
File paths and URLs for content files in storage.....	244
Path specifications for content in file stores.....	244
URL specifications for content files	245
Setting up storage.....	246
Distributed storage setup	247
Setting up blob storage	247
Using DQL to set up blob storage	247
Setting up file store storage areas.....	248
File extensions and the use_extensions attribute	248
Setting base URL	249
Defining file store storage areas as public (Windows only)	249
Using DQL to set up file storage	249
Linked store setup	250
Using DQL to set up linked storage	251
Setting up external storage	253
An example of importing documents stored on a CD-ROM.....	253
Using the Mount method	254
External URL storage setup.....	255
Setting up external free storage	255
Configuring for optimal performance on retrieval.....	255
Setting up content-addressed storage areas	256
Defining storage area retention requirements	257
Setting storage parameters	259
Defining the connection string.....	259
Configuring embedded blob use.....	259
Configuring write attempts in content-addressed storage areas.....	260
Setting clocks and time zones for Centera hosts and Content Server hosts.....	261
Setting up turbo storage.....	261
Providing automatic file extensions	262
Moving content files	262

MIGRATE_CONTENT method	263
Content migration policies	264
Configurable arguments	264
Generated log files	265
Records migration jobs.....	265
Auditing content movement.....	266
Maintenance operations for storage areas.....	266
Changing a storage area's state	267
Determining the state of a storage area.....	267
Moving file store storage areas	267
Enabling forced deletion in content-addressed storage areas	268
Removing orphaned content objects and files	269
Using dmclean	270
Including content in content-addressed storage areas.....	271
Running dmclean using a EXECUTE statement	272
Running dmclean using an Apply method	272
Running dmclean from the operating system prompt	272
Executing the dmclean script.....	273
Using dmfilesan.....	273
dmfilesan arguments	274
Identifying the subdirectories of the scanned storage areas.....	275
Using the -no_index_creation argument.....	276
Using the -force_delete argument	276
Running dmfilesan using an EXECUTE statement.....	276
Running dmfilesan using an Apply method.....	276
Running dmfilesan from the operating system prompt.....	277
The generated script	277
Executing the dmfilesan script	278
Replacing a full distributed storage component	278
Resolving a compromised file store key.....	279
Administering content assignment policies	279
Tracing policy use.....	279
Enabling and disabling assignment policies.....	280
Turning off the policy engine.....	280
Archiving and restoring documents.....	280
How the process works.....	281
The Archive and Restore methods	284
The Archive tool.....	284
Archiving.....	284
Restoring	286
Moving dump files on and off line	286
Archiving content used in multiple documents.....	287
Options for archiving	287
Scheduling archiving	288
Types of request	288
The repository operator	288
Moving the dump file in and out of the archive directory	289
Choosing an archive directory	289
Implementing archiving.....	289
Starting the Archive tool	290
Restoring documents	290
Archiving restored documents	291
Custom restoration	291
Chapter 8 Full-Text Indexing	293
Introducing full-text indexing.....	293
How format objects determine which renditions are indexed.....	294

How content file size determines what is indexed	295
Which languages are indexed	295
How particular characters are handled.....	295
Choosing parts of speech to index.....	296
Overview of the indexing process	297
What comprises a full-text indexing installation	298
Content Server	298
Index agent	298
How the index agent processes queue items	299
Index server and full-text index	300
The full-text index	301
Repository objects and attributes supporting full-text indexing.....	301
Fulltext index object.....	301
FT index agent config object	302
FT engine config object	302
Location objects.....	302
Supporting attributes of other objects.....	302
The a_full_text attribute	302
The fulltext_location attribute.....	303
Entries in the server.ini file	303
The dmfulltext.ini file	303
Managing the index agent and index server	303
Administering the index agent and index server.....	303
Starting and stopping the index agent and index server	304
The index agent at server startup	305
Index agent modes	305
Normal mode	305
Migration mode	305
File mode.....	306
Querying indexes	306
Maintenance operations	306
Turning indexing on and off.....	307
Turning off all indexing	307
Turning off content indexing	307
Index partitioning.....	307
Indexing content in nonindexable formats.....	308
Reindexing a repository	309
Indexing Large Files	310
Enabling thesaurus searching	311
Creating the synonym file	312
Importing the synonym file	312
Logging	314
Log sample	314
Managing the index queue	316
Pointing a repository to a previously-created index.....	316
Obtaining a list of indexable formats.....	317
Tracing full-text query operations	317
Enabling tracing for the index agent.....	317
Chapter 9 Users and Groups	319
User names	320
User privilege levels	321
Basic user privileges	321
Extended user privileges.....	323
Privileged groups	323

Adding users	324
Setting user attributes	325
User privileges	326
Defining the default ACL	326
Setting user_db_name.....	327
Setting default_folder	327
Setting accessible folders.....	327
Setting client capability	327
Creating a new user with DQL	327
Creating a new user with the API	328
Adding multiple users in a single operation	329
LDIF file contents	329
Setting up the file	330
Extended characters in the file	331
Granting and revoking user privileges	331
Superuser privileges and the admingroup group	331
Granting and revoking privileges using DQL	332
Granting and revoking privileges using the API	332
Modifying users	333
Using DQL.....	334
Using the API.....	334
Renaming users.....	334
Deleting users	334
Using DQL.....	335
Using the API.....	335
Deactivating, locking, and reactivating users	336
Deactivating or locking users.....	336
Reactivating users.....	337
Using DQL to change a user's login state.....	337
Adding groups.....	337
Using DQL.....	339
Using the API.....	339
Modifying groups.....	340
Using DQL.....	341
Using the API.....	341
Deleting groups.....	341
Using DQL.....	342
Using the API.....	342
Changing the membership setting of a dynamic group.....	342
Querying groups	343
Obtaining a list of members in a group.....	343
Obtaining a list of groups with a count of the members in each.....	343
Chapter 10 Managing User Authentication	345
Authentication options	345
Default mechanism.....	346
Custom password checking program	346
LDAP directory server	346
Authentication plug-in.....	346
In-line password.....	347
Using the default authentication mechanism	347
UNIX platforms.....	347
Windows platforms	348
Authenticating in domains	348

No-domain required mode.....	348
Domain-required mode.....	349
Determining the repository's authentication mode	349
Converting to domain-required mode.....	349
Using a custom external password checking program	350
Basic steps.....	350
Writing the program	351
Using Windows domain authentication for UNIX users	351
Modifying the dm_check_password program.....	351
Setting auth_protocol.....	353
Setting up the domain controller map	353
Setting the user_source attribute.....	353
Using an LDAP directory server	354
Benefits	354
Constraint.....	355
LDAP authentication options	355
Directory servers and federations	355
How the directory server and a repository are synchronized	356
user_source attribute and the dm_LDAPsynchronization job	357
Integrating an LDAP directory server with a repository	357
Implementing an LDAP directory server.....	358
Defining the set-up values.....	359
Distinguished name and bind type	360
Search bases and filters	360
The secure connection attributes	361
Downloading certutil and the certificate authorities	361
Installing the certificate database and CA certificates	361
Activating the dm_LDAPsynchronization job	362
Building and installing an LDAP-enabled password checking program (UNIX only)	362
Note on using Active Directory	363
Synchronizing users between scheduled synchronization jobs	363
Using multiple LDAP directory servers.....	364
Deleting an LDAP directory server from a repository.....	364
Enabling first-time synchronization rules	364
Binding LDAP users to a different directory server	364
Listing certificates in the certificate database.....	365
Troubleshooting the synchronization job	365
Using authentication plug-ins.....	366
Plug-in scope	366
Identifying a plug-in for use.....	367
Defining a plug-in identifier	367
Using the netegrity plug-in	368
Implementing a custom authentication plug-in.....	368
Writing the authentication plug-in	369
Internationalization	370
Tracing authentication plug-in operations	370
Using an in-line password.....	370
Trusted logins	371
Unified logins.....	371
Managing encrypted passwords	371
Using encryptpass	372
If you do not want to use encrypted passwords	373
Changing an encrypted password.....	373
Limiting authentication attempts	375

Chapter 11	Protecting Repository Objects	377
	Overview of repository security.....	377
	ACLs	378
	Additional security options.....	378
	Application-level control of SysObjects	378
	Dynamic groups.....	379
	Folder security	379
	User privileges	380
	Table permits	380
	Turning repository security on and off	380
	Turning folder security on and off	381
	Changing folder security.....	382
	Setting the default permission level for application-level control of SysObjects.....	383
	Object-level permissions	383
	Basic permissions	383
	Note on the Relate permit level.....	384
	Extended permissions	385
	Viewing extended permissions	386
	Managing ACLs	387
	The ACL object type	388
	Access control entries.....	388
	AccessPermission and ExtendedPermission entries.....	389
	AccessRestriction and ExtendedRestriction entries.....	389
	AccessRestriction entries.....	389
	ExtendedRestriction entries.....	390
	Storage in the ACL	390
	ApplicationPermission entries.....	390
	ApplicationRestriction entries	391
	RequiredGroup entries	391
	RequiredGroupSet entries.....	392
	How ACL entries are evaluated.....	392
	Evaluation for non-owners and non-superusers.....	393
	How access is evaluated for object owners and superusers	393
	Access evaluation for an object's owner	393
	Evaluating a Superuser's permissions	395
	Resolving multiple entries for a user	395
	Disabling ACL restrictive entries	396
	External and internal ACLs	396
	ACL names	397
	System, public, and private ACLs	397
	Template ACLs.....	398
	Creating ACLs	398
	How ACLs and objects are connected	399
	The default ACLs	400
	Creating default ACLs	401
	Assigning a default ACL to an object	401
	Identifying the default ACL for use	401
	Modifying an ACL.....	402
	Adding entries	402
	Removing entries.....	403
	Destroying an ACL.....	403
	Removing unreferenced external ACLs.....	403
	Removing unreferenced internal ACLs	403
	Table permits	404
	Setting table permits	405

Table permits and object-level permissions	406
Table permits and dump and load operations	406
Auditing	406
What events are auditable	407
Auditing attributes	407
Audit trails.....	408
Default auditing	408
Auditing system events.....	409
Auditing application events	409
Signing audit trail entries	410
Conflicting registration resolution	411
Stopping auditing.....	412
Viewing audit trails	412
Querying and retrieving audit trail entries	413
Interpreting audit trails of API, workflow, and lifecycle events	413
Audit trail attributes with a common purpose	413
Attributes available for varied purposes.....	413
Use in API audit trails.....	414
Use in lifecycle audit trails	419
Use in workflow audit trails	420
Interpreting ACL and group audit trails.....	427
Verifying signed audit trail entries	429
Removing audit trail entries	430
Auditing in a distributed environment.....	430
Implementing signature support	431
Customizing electronic signatures	431
Customizing the default functionality	432
Adding or removing properties on the page	434
Changing the property delimiters	435
Configuring the appearance of the page	435
Defining separate templates for different document types	436
Configuring the number of allowed signatures and signature positioning	437
Creating custom signature creation methods and associated signature page templates	438
Creating custom signature-creation methods.....	438
Creating custom signature page templates	440
Tracing electronic signature operations	440
Supporting digital signatures	440
Customizing simple signoffs	441
Customizing the signature validation program.....	442
Registering for notification.....	442
Querying the audit trail for signoffs.....	442
Managing the encryption keys.....	443
The AEK	443
Sharing the AEK or passphrase	444
The AEK and distributed sites	444
Backing up the AEK.....	444
Repository encryption keys	445
Encryption utilities	445
Using dm_crypto_boot	446
Troubleshooting with dm_crypto_create	447
Changing a passphrase	447
Managing the login ticket key.....	449
Exporting and importing a login ticket key.....	449
Resetting a login ticket key	449
Configuring a repository's trusted repositories	449

Configuring login ticket use	450
Configuring the default login ticket timeout	450
Restricting a Superuser's use of global tickets	451
Revoking tickets for a specific repository	451
Configuring application access control token use	451
Enabling AAC token use by a server	452
Enabling token retrieval by the client library	452
Generating tokens for storage	453
Naming the output file	455
Storing tokens generated by dmtkgen	455
Chapter 12 Tools and Tracing	457
Essential tool concepts	458
How tools are implemented	460
Standard arguments	461
The QUEUEPERSON argument	461
The window interval	462
Reports and trace log files	462
Reports	462
Storage	463
Trace log files	463
Storage	464
Email messages	464
Activating and scheduling administration tools	465
Activation	465
Defining job schedules	465
Running administration jobs on demand	465
Archive	466
Audit Management	468
Consistency Checker	471
Running the job from a command line	471
Content Replication	477
Content Warning	480
Create Full-Text Events	482
Arguments	483
Report sample	486
Data Dictionary Publisher	487
Database Space Warning	489
Dm_LDAPsynchronization	491
Executing dm_LDAPsynchronization manually	494
Dmclean	494
Dmfilescan	499
File Report	504
Group Rename	509
Index Agent Startup	510
Log Purge	510
Queue Management	514
Remove Expired Retention Objects	517
Rendition Manager	520
State of the Repository Report	525

Swap Info	529
ToolSetup.....	530
Update Statistics	530
User Chg Home Db	533
User Rename.....	534
Version Management	536
Tool maintenance and troubleshooting.....	540
Changing the default settings	540
Using DQL.....	540
Using API.....	541
Using the tool trace log files	541
Viewing the tool reports.....	541
Tracing	541
Types of tracing	542
Server tracing.....	542
Server start-up tracing options.....	543
Changing server tracing after start-up.....	543
Examples of server tracing	544
Session tracing	546
Severity levels	546
Tracing severity level 10	547
Tracing severity level 11	547
Examples of session tracing.....	548
Client tracing	549
Appendix A Consistency Checks	551
User and group checks.....	552
ACL checks.....	553
SysObject checks	554
Folder and cabinet checks	555
Document checks	556
Content object checks.....	556
Workflow checks	557
Object type checks.....	558
Data dictionary checks.....	558
Lifecycle checks.....	559
Object type index checks	560
Method object consistency checks.....	561
Appendix B IAPI and IDQL	563
Using IAPI.....	563
Starting IAPI	563
IAPI commands.....	566
Entering method calls	568
Entering comments.....	569
Quitting an IAPI session	569
Using IDQL.....	569
Starting IDQL.....	569
The IDQL commands.....	572
Entering queries	573
Clearing the buffer.....	573

	Entering comments.....	573
	Stopping IDQL.....	574
Appendix C	Archiving scripts	575
	The dql Script.....	575
	The dm_archive_start script	576
	The post_archive script	577
	The pre_restore script	581
	The post_restore script.....	585
Appendix D	High-Availability Support Scripts	589
	Monitoring scripts	589
	Processes not requiring a script	591
Appendix E	Populating and Publishing the Data Dictionary	593
	Populating the data dictionary.....	593
	Data dictionary population script	594
	The initialization file	594
	The data files.....	595
	File structure	595
	Summary of settable data dictionary attributes	596
	Setting foreign keys	601
	Examples of data file entries	602
	Executing the script	605
	Populating a Japanese locale on a Korean host or a Korean locale on a Japanese host	605
	Default files provided by EMC Documentum	606
	Using DQL statements	606
	Publishing the data dictionary information	607
	The data dictionary publisher job	607
	The Publish_dd method	608
	The PUBLISH key phrase.....	608
Appendix F	Supported and Unsupported Formats for Full-Text Indexing	609
Appendix G	Supported Languages for Full-Text Indexing	617

List of Figures

Figure 7-1.	Relationship of content assignment policies and object types	240
Figure 7-2.	Relationship between a file store object and a location.....	248
Figure 7-3.	The objects that define a linked store storage area	251
Figure 7-4.	The archive and restore process	283

List of Tables

Table 2-1.	Users created during repository configuration.....	41
Table 2-2.	Groups created during repository configuration.....	41
Table 2-3.	security_type attribute settings for location objects	64
Table 2-4.	Interaction of attributes determining package control behavior.....	81
Table 3-1.	Default code page settings by host machine locale	87
Table 3-2.	Server.ini SERVER_STARTUP section keys.....	89
Table 3-3.	Server.ini DOCBROKER_PROJECTION_TARGET keys	92
Table 3-4.	Server.ini FUNCTION_SPECIFIC_STORAGE keys	93
Table 3-5.	Server.ini TYPE_SPECIFIC_STORAGE keys	93
Table 3-6.	Server.ini FUNCTION_EXTENT_SIZE keys	93
Table 3-7.	Server.ini TYPE_EXTENT_SIZE keys	94
Table 4-1.	Standard arguments passed to jobs	149
Table 4-2.	dcf_edit utility arguments	154
Table 4-3.	Entries in a job sequence statusreport.....	156
Table 4-4.	dm_run_dependent_jobs arguments.....	157
Table 5-1.	Scope of changes to client and session configuration parameters.....	173
Table 5-2.	dmcl.ini DMAPI_CONFIGURATION keys.....	175
Table 5-3.	Configuration settings for uUse of client local area and server common area	184
Table 5-4.	Interaction of persistent caching dmcl.ini keys and methods.....	193
Table 7-1.	Summary of storage area configuration options	235
Table 7-2.	Examples of URLs and formats	255
Table 7-3.	dmclean arguments	270
Table 7-4.	Arguments to the dmfilesan utility	274
Table 8-1.	Syntax of ImportDictionary.py script	313
Table 9-1.	Basic user privileges	321
Table 9-2.	Extended user privileges	323
Table 9-3.	Privileged groups	324
Table 9-4.	Security attributes for new users.....	326
Table 9-5.	Default values for new users	330
Table 10-1.	Netegrity plug-in files.....	368
Table 10-2.	Password files encrypted by default.....	372
Table 11-1.	Repository security settings.....	380
Table 11-2.	Object-level permissions	384
Table 11-3.	Permitted operations for object-level pPermissions	384
Table 11-4.	Extended object-level permissions	385
Table 11-5.	The extended permission computed attributes.....	386

Table 11-6.	Table permits	404
Table 11-7.	Usage of generic attributes by API events.....	414
Table 11-8.	Usage of generic attributes by lifecycle events	420
Table 11-9.	Usage of generic attributes by workflow events	420
Table 11-10.	System-defined document properties for the signature page template	433
Table 11-11.	System-defined signature properties for the signature page template	434
Table 11-12.	Parameters passed by Addesignature to the signature-creation method	439
Table 11-13.	Interaction of dm_crypto_change_passphrase arguments.....	448
Table 11-14.	dmtkgen utility arguments.....	454
Table 12-1.	EMC Documentum tool suite	458
Table 12-2.	Standard arguments passed to jobs	461
Table 12-3.	Archive arguments	466
Table 12-4.	Audit Management arguments.....	469
Table 12-5.	Content Replication arguments	478
Table 12-6.	Content Warning arguments	481
Table 12-7.	Create Full-Text Events arguments.....	484
Table 12-8.	Data Dictionary Publisher argument	488
Table 12-9.	Database Space Warning arguments	489
Table 12-10.	dm_LDAPsynchronization arguments.....	492
Table 12-11.	Dmclean arguments.....	495
Table 12-12.	Dmfilesan arguments	500
Table 12-13.	File Report arguments.....	507
Table 12-14.	Files deleted by the Log Purge tool	510
Table 12-15.	Log Purge arguments.....	511
Table 12-16.	Queue Management arguments.....	515
Table 12-17.	Remove Expired Retention Objects arguments.....	518
Table 12-18.	Rendition Management arguments	520
Table 12-19.	State of the Repository arguments.....	525
Table 12-20.	Swap Info arguments	529
Table 12-21.	Update Statistics arguments	531
Table 12-22.	Version Management arguments	537
Table 12-23.	Server start-up trace options.....	543
Table 12-24.	Trace level severity values	546
Table A-1.	Consistency checks for users and groups.....	552
Table A-2.	Consistency checks for ACLs	553
Table A-3.	Consistency checks for SysObjects	554
Table A-4.	Consistency checks for folders and cabinets	555
Table A-5.	Consistency checks for documents.....	556
Table A-6.	Consistency checks for content objects	557
Table A-7.	Consistency checks for workflows	557
Table A-8.	Consistency checks for object types	558
Table A-9.	Consistency checks for the data dictionary	559

Table A-10.	Consistency checks for lifecycles.....	560
Table A-11.	Consistency checks for object type indexes	560
Table A-12.	Consistency checks for method objects.....	561
Table B-1.	IAPI command line arguments	564
Table B-2.	IAPI commands.....	567
Table B-3.	IDQL command line arguments.....	570
Table B-4.	IDQL commands	572
Table D-1.	Monitoring Scripts.....	589
Table E-1.	Settable data dictionary attributes using population script	597
Table F-1.	Supported document formats	609
Table F-2.	Unsupported document formats.....	615
Table G-1.	Supported languages	617

Preface

This manual contains information, instructions, and procedures for the normal system administration tasks of a Documentum installation. It gives you an overview of the system's configuration and some guidelines for making your own configuration decisions. It provides instructions for changing the configurations of repositories, Content Servers, clients, and sessions as well as instructions for routine maintenance. It also describes connection brokers, full-text indexing administration, managing content storage area, and repository security.

Intended audience

This manual is written for system and repository administrators. The system administrator is the person who generally installs and owns the Documentum installation. Repository administrators are the users who own and are responsible for one or more repositories. Readers are expected to be familiar with the general principles of client-server architecture and networking. In addition, they should know and understand the Windows and UNIX operating systems.

Conventions

This manual uses the following conventions in the syntax descriptions and examples.

Syntax conventions

Convention	Identifies
<i>italics</i>	A variable for which you must provide a value.
[] square brackets	An optional argument that may be included only once
{ } curly braces	An optional argument that may be included multiple times

Terminology changes

Two common terms are changed in 5.3 and later documentation:

- Docbases are now called repositories, except where the term “docbase” is used in the name of an object or attribute (for example, docbase config object).
- DocBrokers are now called connection brokers.

Revision history

The following changes have been made to this document.

Revision history

Revision Date	Description
September 2005	Initial publication

Introduction

This chapter is an introduction to administering the Documentum Content Server™. It includes the following topics:

- [Essential concepts, page 28](#), which briefly outlines basic Content Server and repository configuration choices
- [Administration tasks, page 30](#), which briefly describes the typical administration tasks for individual repositories
- [User privilege requirements for administration tasks, page 32](#), which describes the user privilege level required for most administration tasks
- [Administration interfaces, page 32](#), which describes the user interfaces you can use to perform administration tasks
 - [Starting Documentum Administrator, page 32](#), which contains instructions for connecting to a repository using Documentum Administrator.
 - [Using the API for administrative tasks, page 33](#), which describes the options for performing administration tasks using the API directly
 - [Using DQL for administrative tasks, page 33](#), which describes the options for performing administration tasks through DQL statements
- [Documentum tool suite, page 34](#), which introduces the suite of administration tools provided with Content Server
- [Administration methods, page 34](#), which introduces the administration methods
- [The dm_error utility, page 34](#), which describes the dm_error utility used to obtain additional information about an error.
- [Viewing connected users, page 35](#), which contains instructions for determining whether users are currently connected to a server
- [Where to look for more information, page 35](#)

The *Content Server Administrator's Guide* contains information and procedures for administration tasks that apply to individual repositories and non-distributed storage areas. Administration tasks associated with distributed configurations are described in the *Documentum Distributed Configuration Guide*.

Essential concepts

A Documentum installation has many aspects that you can arrange to suit your requirements. For instance, you can choose how many repositories to create, how many servers for each repository, where to put the content storage areas, whether to have separate tablespaces or segments for type indexes, and so forth. Most of these decisions can be changed. That is, you can choose one alternative when you first set up your installation and then modify it later.

In fact, we recommend that your first installation be as simple as possible to ensure that you get up and running as quickly as possible. After you are familiar with Documentum, you can modify the installation if necessary.

What comprises an installation?

In a general sense, a Documentum installation consists of one or more repositories, the servers that access the repositories, and the clients that access the repositories through the servers. However, from the viewpoint of a system administrator, the installation also includes its supporting parts: the machines, the networking software, the file-sharing software, the relational database, and so forth. The requirements for these parts—for example, which machines, operating systems, and databases Documentum can run on—are found in the *Content Server Release Notes*. This chapter provides some guidelines for deciding what the configuration of your installation will be.

Configuration choices

You have a number of choices for your installation. The choices fall into these broad categories:

- Overall system architecture

For example, before you install Documentum, you must decide if you want to put Content Server and the RDBMS on the same machine or different machines and where to locate the full-text indexing software. You should also determine how many servers you want for any particular repository. repositories that are accessed more frequently or by users at different geographical sites generally have multiple servers. The *Content Server Installation Guide*, *Content Server Full-Text Indexing Installation Guide*, and *Distributed Configuration Guide* contain information on system architecture. [Chapter 3, Servers](#) and [Chapter 2, Content Repositories](#) in this manual contain information on configuring servers and repositories.

- Server configuration

Servers can vary widely in their configuration. When you start a Content Server, its configuration is taken from parameters set in its server.ini file (an initialization file) and from the values in its server config object. For example, through the server config object, you can control the number of concurrent sessions a server accepts, the server and client cache sizes, what user-defined object types the server caches on startup, and which storage areas it can use. The server.ini file lets you define the server's proximity to connection brokers (and consequently, to clients). [Chapter 3, Servers](#), contains more details about your choices when configuring a server.

- Repository configuration

Choices specific to a repository include what types of storage areas will you use and how many, whether to enforce security for SysObject operations, whether to implement full-text indexes, and so forth. [Chapter 2, Content Repositories](#), contains information on the choices available to you when configuring a repository.

- Client configuration

From the client side, you can determine how you want a particular client to share files with the server, how you want it to display dates, and which connection broker you want it to use. [Chapter 5, Clients and Sessions](#), contains information on client configuration.

If you choose to use the basic installation that is in place after you install Content Server, you only have to decide where you want to put the installation. If you want to modify the basic installation, then the number and scope of the decisions you make will depend on what modifications you want to make.

Configuration objects

In each repository, there are a variety of objects that, taken together, define your configuration. For example, the objects include:

- Server config object
- Docbase config object
- Location objects
- Mount point objects
- Storage objects
- Format objects
- Method objects

As you make choices about how you want to configure your installation and repositories, you will modify these objects or add new ones to implement your decisions.

Administration tasks

After Content Server is installed and running, typical system administration tasks for a repository or installation include:

- Modifying the configuration of the server, connection broker, or client sessions

[Chapter 3, Servers](#), [Chapter 2, Content Repositories](#), and [Chapter 5, Clients and Sessions](#) contain information on these tasks. The *Distributed Configuration Guide* contains instructions on creating distributed configurations and federations.

- Creating new repositories

Installing Content Server creates one repository, one server, and one connection broker. It is probable that your enterprise will want additional repositories. Instructions for creating new repositories are found in [Chapter 2, Content Repositories](#).

- Maintaining content repositories

Businesses grow and change continuously. This is reflected in changing repository requirements. Repository maintenance tasks keep repositories synchronized with business needs by adding objects as needed and removing unwanted or unneeded objects. These tasks include:

- Creating new object types, new format objects, and new alias set objects.

These objects typically support repository users and applications that access the repository. [Chapter 2, Content Repositories](#), describes how to create these objects.

- Creating new method objects, new jobs, and new configuration objects such as locations and mount point objects.

Method objects and jobs typically support system administrators by allowing administrators to automate some administration tasks. They can also be used to automate some business processes, such as report generation. Location and mount point objects support the repository architecture. [Chapter 4, Methods and Jobs](#), describes how to create method and job objects. [Chapter 2, Content Repositories](#), describes how to create location and mount point objects.

- Cleaning up repositories

Some user operations leave orphaned objects in the repository. For example, when users delete documents, the associated content files are orphaned in the storage areas. Cleaning up orphaned objects is an important system administration task. [Chapter 2, Content Repositories](#), describes how to clean up a repository.

- Maintaining repository consistency

The Consistency Checker administration tool provides important information on repository consistency. [Appendix A, Consistency Checks](#), describes each check run by the tool.

- Configuring, starting, and shutting down servers

[Chapter 3, Servers](#), includes procedures for server administration.

- Changing session configurations

[Chapter 5, Clients and Sessions](#), includes procedures for changing session configurations and managing local areas.

- Maintaining connection brokers

The connection broker is Documentum's name server. It provides Content Server connection information to client sessions. [Chapter 6, Connection Brokers](#), describes how to configure the connection broker behavior and how to stop and start a connection broker.

- Managing content storage areas and content files

The basic installation has one storage area for content files. Typically, you will add more storage areas immediately after installation or later, as the repository grows. [Chapter 7, Content Management](#), describes the types of content storage areas you can use and includes procedures for setting up and maintaining them. The chapter also includes instructions for archiving and restoring content files.

- Administering full-text indexes

[Chapter 8, Full-Text Indexing](#), describes full-text indexes and includes procedures for setting up and maintaining full-text indexes.

- Managing users and groups

[Chapter 9, Users and Groups](#), contains procedures for adding, changing, and removing local users and groups.

- Managing security

Documentum's security features include ACLs, object-level permissions, table permits, user privilege levels, and auditing and tracing capabilities.

[Chapter 11, Protecting Repository Objects](#), describes Documentum's implementation of ACLs, object-level permissions, and table permits. The chapter also describes the auditing capabilities. It includes procedures for creating and maintaining ACLs (which enforce object-level permissions), setting table permits, and for turning auditing on and off.

User authentication is described in [Chapter 10, Managing User Authentication](#).

User privileges are described in [Chapter 9, Users and Groups](#).

Tracing capabilities are described in [Chapter 12, Tools and Tracing](#).

User privilege requirements for administration tasks

Most of the system administration tasks in this manual require at least the Sysadmin user privilege. Some, such as deleting audit trail entries, require one of the extended privileges. The instructions for a particular procedure inform you what level of user privileges is required.

When you connect to a repository to perform an administrative task, log in as a user with at least the minimum required level of user privileges needed for the procedure.

Administration interfaces

The primary user interface for administration tasks is Documentum Administrator. Documentum Administrator is a Web-based interface that lets you monitor, administer, configure, and maintain Documentum repositories throughout the enterprise (both local and federated) from any system running a Web browser. Documentum Administrator also provides easy access to the Documentum System Administration Tool suite and the administration methods.

Documentum Administrator is sold separately; it is not included with Content Server. For instructions on starting Documentum Administrator and connecting to a repository, refer to [Starting Documentum Administrator, page 32](#). (Instructions for installing Documentum Administrator are found in the *WDK and Applications Installation Guide*.)

You can also use the DMCL API (application programming interface) or DQL (Document Query Language) directly, to perform some administrative tasks. To use the API, you enter methods through Documentum Administrator or use the IAPIutility. To use DQL, you can enter DQL statements through Documentum Administrator or use the IDQL utility. [Appendix B, IAPI and IDQL](#), describes how to use these utilities.

Starting Documentum Administrator

You can start Documentum Administrator from Internet Explorer or Netscape Communicator. Use the URL provided by your system or repository administrator to access Documentum Administrator.

Documentum Administrator allows you to change the connection broker in use. You can also identify a particular server as the target of your repository connection request. If you need help to identify a different connection broker or to connect to a repository, refer to Documentum Administrator online help.

Using the Content Server Manager on Windows

The Content Server Manager is an administration tool added during Server installation. You can use the Content Server Manager to:

- Change your server configuration by editing the server.ini file
- Uninstall a repository
- Start the IAPI or IDQL utilities
- Change how you start a connection broker
- Display connection broker log files
- Invoke the Setup program to install or uninstall additional server components
- Invoke the Microsoft Performance Monitor tool
- Create a configuration summary report for troubleshooting

Using the API for administrative tasks

There are two ways to issue API methods to administer the server:

- Documentum Administrator
- The IAPI utility

To use Documentum Administrator, refer to the Documentum Administrator online help.

The IAPI utility is a command-line interface that lets you connect to a repository and execute API methods. The utility is provided with Content Server. For information about using IAPI, refer to [Using IAPI, page 563](#).

Using DQL for administrative tasks

You can use DQL statements to perform some administration tasks. There are three ways to issue DQL statements:

- Documentum Administrator
- The IDQL utility
- The Execquery, Readquery, Query_cmd, and Cachequery API methods

To use Documentum Administrator, refer to the Documentum Administrator online help.

The IDQL utility is a command-line interface that lets you connect to a repository and execute DQL statements. The utility is provided with Content Server. For information about using IDQL, refer to [Using IDQL, page 569](#).

The Execquery, Readquery, Query_cmd, and Cachequery API methods accept DQL statements as arguments.

Documentum tool suite

Documentum provides a suite of administration tools with Content Server. These tools automate such tasks as:

- Removing unwanted renditions, content files, and log files
- Monitoring space in the storage areas and database tables
- Managing full-text indexes

The tools are implemented as jobs and most are installed in the inactive state. For a description of each tool, refer to [Chapter 12, Tools and Tracing](#). That chapter also contains instructions for activating the tools, resetting their schedules, and viewing their reports and log files.

Administration methods

Documentum provides a variety of administration methods that you can use to perform many administration tasks, such as managing full-text indexes, managing content files, or monitoring sessions. You can run the methods from Documentum Administrator or using the Apply method or the DQL EXECUTE statement. For complete information about these methods, refer to [Chapter 3, Administration Methods](#), in the *Content Server DQL Reference Manual*.

The dm_error utility

The dm_error utility is a utility that returns information about a specified error. The utility is run from the operating system prompt. The command line is:

```
dm_error <error_code>
```

The *error_code* is the abbreviated text that describes the error. For example, DM_SERVER_E_NO_MTHDSVR_BINARY.

The output of the utility lists the error and a description of its cause and actions to take. Here is a sample of the output:

```
[DM_SERVER_E_NO_MTHDSVR_BINARY]
"%s: Method server binary is not accessible."
CAUSE: The method server binary "mthdsvr" is not under
```

```
$DM_HOME/bin or it does not have the proper permission.  
ACTION: Make sure method server binary "mthdsvr" is under  
$DM_HOME/bin and set execution permission for it.
```

Viewing connected users

Before you begin a system administration procedure, you may want to determine whether any users are currently logged in to the system. Use the Sessions page in Documentum Administrator to obtain this information. For further instructions, refer to the Documentum Administrator online help.

For more information about clients and sessions, refer to [Chapter 5, Clients and Sessions](#).

Where to look for more information

This book provides instructions for tasks that affect a single repository. Information and instructions for tasks that affect federations or distributed storage areas are found in the *Documentum Distributed Configuration Guide*.

For information about installing the Content Server, refer to *Content Server Installation Guide*.

For detailed information about Documentum objects, DQL statements, and API method calls, refer to the reference manuals:

- *EMC Documentum Object Reference*
- *Content Server DQL Reference*
- *Content Server API Reference*

Content Server Fundamentals describes the basic features of the Content Server and the repository. It explains the data model, session and transaction management, content management, and the behavior and implementation of features such as workflows and lifecycles.

Content Repositories

This chapter contains the following information about repositories:

- [Essential concepts, page 38](#), which discusses how repositories and servers are connected and repository configuration
- [Adding additional repositories, page 39](#), which contains instructions for creating repositories
- [Managing cabinets and folders, page 43](#), which describes the how to create, modify, and delete cabinets and folders
- [Setting the dd_locales attribute, page 45](#), which describes how to set this docbase config attribute
- [Manipulating type indexes, page 45](#), which describes how to create, move, and delete object type indexes
- [Alternate locations for object-type tables on Oracle and DB2, page 46](#), which describes alternative configuration choices for object type tables
- [Dumping and loading a repository, page 46](#), which contains instructions for dumping and loading a repository
- [Creating location and mount point objects, page 62](#), which describes how to create location and mount point objects
- [Format objects, page 66](#), which describes how to create, query, and manage format objects
- [Alias sets, page 72](#), which describes how to create, modify, and destroy alias sets
- [Working with object types, page 73](#), which describes how to create, modify, and delete user-defined object types
- [Cleaning up repositories, page 77](#), which describes how to use Documentum utilities to clean up repositories
- [Maintaining query performance, page 80](#), which discusses RDBMS statistics maintenance to maintain optimal query performance
- [Configuring repository-level package name control, page 80](#), which describes how to enable or disable workflow package name control.
- [Configuring a repository printer on Windows, page 81](#)

Essential concepts

Repositories are comprised of object type tables, type indexes, and content files. The type tables and type indexes are tables in an underlying relational database. The content files are typically stored in directories on disks in your installation. However, content files can also be stored in the database or on external storage devices.

Full-text indexes are associated with a particular repository. They enable rapid searching for text strings in content files and attribute values.

Users access repositories through servers. The servers receive queries from clients in the form of API methods or DQL statements and make the actual call to the underlying RDBMS or the file directories. Every repository must have at least one active Content Server. If a repository does not have an active server, then users cannot access that repository.

How are repositories and servers connected?

Information that you provide about the repository when you create it is used to build a server startup file. The startup file is executed whenever you start the repository's server. The information in that file binds the Content Server to the repository.

The default installation starts one Content Server for a repository. However, you can start multiple servers for the same repository. (Refer to [Starting additional servers](#), page 110, for instructions.) If a repository is very active, serving many users, or its users are widely spread geographically, having multiple servers can provide load balancing and enhance performance. Or perhaps you want to dedicate one server to a particular application or group of users and have other servers available to everyone. Multiple servers give you those options.

Repository configuration

A repository's operating configuration is defined by its doabase config object, which resides in the repository. This object is created when you create the repository. The attributes in the doabase config object record information such as the:

- Name of the underlying RDBMS
- Security level for the repository
- Whether folder security is turned on
- Macintosh access protocol
- Alternate tablespace for the type indexes

The repository's structural configuration, that is, where its content and index files are, what file formats it recognizes, and so forth, is defined in the repository itself—by objects. The objects that define the repository structurally are:

- Location Objects

A location object defines the location of a particular file or directory for the Content Server.

- Storage Objects

Storage objects define content storage areas. There are several types of storage areas, each represented by a different storage object type. A storage area object is paired with a location object to define content storage areas in a repository. For more information about the types of storage areas, refer to [Chapter 7, Content Management](#).

- Mount Point Objects

If a number of locations reside under one upper-level directory, and all the locations must be visible to the clients, use a mount point object to define the location of the upper-level directory. Using mount point objects alleviates the need to mount multiple directories on clients.

- Format Objects

Format objects define file formats. A server only recognizes a file format if the format has a format object in the repository.

- Fulltext Index Objects

Fulltext index objects represent full-text indexes. If you index some or all of the content files in your repository, you have fulltext index objects in the repository. The basic installation is configured for full-text indexing. (For information about managing full-text indexes and directories, refer to [Chapter 8, Full-Text Indexing](#).)

You must have Sysadmin or Superuser user privileges to change a repository's doabase config object.

Adding additional repositories

Note: This procedure describes the basic steps for adding another repository to an existing installation. If you are setting up the initial Documentum Content Server installation, refer to the *Content Server Installation Guide* for instructions.

When you create a new repository, the installation wizard:

- Asks you several questions about the new repository, such as its ID and name, and uses your answers to build the Content Server start-up files.

Note: A repository name must consist of ASCII characters and be less than or equal to 32 characters in length. The name docu is reserved by Documentum.

- Creates the database account for the new repository if you wish.

Note: For DB2, a database account is not needed. (However, you must grant an existing account certain authorities or privileges to access the database. For example, the installation owner must have DBADM authority. Refer to the installation manual for more information.)

- Runs the Content Server startup files to start Content Server.
- Executes the headstart.ebs file to populate your repository with a basic set of configuration-related objects, such as the basic location objects, format objects, and method objects.

You must be logged in as the installation owner to create a new repository. However, you can designate any user as the repository owner.

Many sites place the storage areas, the share directory, or both on disks that are separate from the disk that holds the Documentum installation executables (the product and dba directories). If your site has done this, you must edit entries in headstart.ebs script to point to the correct directories before executing the script.

Procedure

This procedure outlines the basic steps to add a repository to an existing Documentum installation.

To create an additional repository:

1. Read the information in *Content Server Installation Guide* concerning repository configuration and the prerequisite decisions and actions.
2. Log in as the installation owner.
3. Change to the `$DOCUMENTUM/product/version/install` directory.
4. Follow the procedure for Configuring the Repository in the installation manual (*Content Server Installation Guide*). Start with step 2.

Configuring the new repository for use with Media Transformation Services

If your site has Documentum Media Transformation Services installed and you intend to store rich media documents in the new repository, install (or reinstall) Media Transformation Services on the Content Server host machine.

Each repository must have a dedicated Media Server, and the `base_url` attribute for the rich media storage areas in the repository must be properly set. A Thumbnail Server can service multiple repositories. However, if you rerun the Thumbnail Server installation procedure, it automatically sets the `base_url` attribute correctly for the new repository. (Note that the new repository must be running before you rerun the Thumbnail Server install.)

Refer to *Media Transformation Services Installation Guide* for instructions on installing Media Servers and Thumbnail Servers.

Contents of new repositories

A new repository is not actually empty. The installation program and the scripts that run during repository configuration automatically create a variety of objects, such as cabinets, configuration objects, users, and groups.

[Table 2-1, page 41](#), lists the users created during repository configuration and the privileges each has by default.

Table 2-1. Users created during repository configuration

User	User Privileges	Extended User Privileges
<i>repository_owner</i>	16 (Superuser)	None
<i>installation_owner</i>	16 (Superuser)	None
dm_autorender_win31	8 (Sysadmin)	None
dm_autorender_mac	8 (Sysadmin)	None
dm_mediaserver	8 (Sysadmin)	None
dm_fulltext_index_user	16 (Superuser)	None

[Table 2-2, page 41](#), lists the groups created during repository configuration.

Table 2-2. Groups created during repository configuration

Group	Members
admingroup	<i>repository_owner</i> <i>installation_owner</i>
dm_browse_all	Members of this group can browse any object in the repository. This group has no default members.

Group	Members
dm_browse_all_dynamic	<p>This is a dynamic group whose members can browse any object in the repository.</p> <p>This group has no default members.</p>
dm_fulltext_admin	<p><i>installation_owner</i></p> <p>This is a group for users who can modify the full-text indexing configuration objects.</p>
dm_retention_managers	<p>Members of this group can own retainer objects (representing retention policies) and add and remove a retainer from any SysObject.</p> <p>This is a dynamic group.</p> <p>This group has no default members.</p>
dm_retention_users	<p>Members of this group can add retainers (retention policies) to SysObjects.</p> <p>This is a dynamic group.</p> <p>This group has no default members.</p>
dm_superuser	<p>Members of this group are treated as superusers in the repository.</p> <p>This group has no default members.</p>
dm_superuser_dynamic	<p>A dynamic group whose members are treated as superusers in the repository.</p> <p>This group has no default members.</p>
docu	<p><i>repository_owner</i> <i>installation_owner</i> dm_autorender_win32 dm_autorender_mac dm_mediaserver</p>
queue_admin	<p>None</p> <p>This is a role group supporting the queue management feature of Business Process Manager.</p>

Group	Members
queue_manager	queue_admin group This is a role group supporting the queue management feature of Business Process Manager.
queue_processor	queue_manager group This is a role group supporting the queue management feature of Business Process Manager.
process_report_admin	queue_admin group This is a role group supporting the queue management feature of Business Process Manager.

Managing cabinets and folders

The objects in a repository are organized using cabinets and folders. Cabinets provide the highest level of organization. Every document and folder must reside in a cabinet. There is no limit to the number of cabinets in a repository or the number of objects in a cabinet.

Folders can be placed inside cabinets or other folders. Documents and other objects are stored inside folders or directly inside cabinets. Placing related documents within folders and related folders within cabinets helps organize them so that you or someone else can find them later. For example, you can easily organize a repository's cabinets or folders by project or department.

Users place objects into folders and cabinets using Desktop Client, Intranet Client, or a customized client interface. Applications use the Link method to perform those operations. (Refer to [Link](#), page 297, in the *Content Server API Reference Manual* for a complete description of the method.) Documentum Administrator also allows you to link objects to a folder or cabinet.

Public and private cabinets

A cabinet's `is_private` attribute indicates whether the cabinet is private or public. If set to `TRUE`, the cabinet is private. If set to `FALSE`, the cabinet is public. The default is `FALSE`. This attribute is not used by Content Server for security or any other use. It is intended for use by client applications. For example, Documentum Desktop Client keeps private cabinets invisible to all but their owners.

Home cabinets

Typically, every user has a home cabinet. The home cabinet is where users store personal documents, folders, and Smart Lists. A user may have a home cabinet in every repository. If your repositories are federated, global users typically have only one home cabinet, which is found in their home repository. (Home cabinets and home repositories are defined in the user object.)

Creating folders and cabinets

Any user can create folders. Creating cabinets requires the Sysadmin, Superuser, or Create Cabinet privileges.

It is easiest to create folders and cabinets using Documentum Administrator. (If you need instructions, refer to Documentum Administrator online help.) You can also use DQL or API methods.

To create a cabinet or folder using DQL, use the CREATE...OBJECT method.

To use the API, you must issue a series of methods:

Create—Create a new instance of the folder or cabinet object type

Set—Set its attributes (including name)

Save—Store the object

For a complete list of cabinet and folder attributes, refer to [Cabinet, page 107](#), and [Folder, page 246](#), of the *EMC Documentum Object Reference Manual*.

Changing and deleting folders and cabinets

You can modify or delete cabinets and folders using Documentum Administrator, DQL, or the API. Using Documentum Administrator allows you to:

- View and edit cabinet and folder attributes
- Add or remove links to a folder
- Delete a cabinet or folder

In DQL, use an UPDATE...OBJECT or DELETE...OBJECT statement.

If you use the API, you must fetch the object and then set its attributes and save the object. You cannot checkout a folder or cabinet.

Changing a cabinet or folder's attributes requires at least Write permission on the object. Adding or removing links for a folder also requires Write permission on the folder.

Deleting a folder requires Delete permission for the folder. Deleting a cabinet requires Superuser, Sysadmin, or Create Cabinet privilege.

Setting the dd_locales attribute

The dd_locales attribute in the docbase config object records the data dictionary locales recognized by the Content Server. This attribute is set automatically when you add a locale to the data dictionary using the dd_populate.ebs script. (Refer to [Populating the data dictionary, page 593](#), for information about the script.)

You must have superuser privileges to set this attribute manually, and you must reinitialize the server after setting the attribute to make the change visible to the server.

The locale value is stored in the attribute in its preferred format. For example, if you set the value to fr_ca for French Canadian, the value is stored as fr_CA.

Manipulating type indexes

Indexes on the object type tables in the RDBMS enhance the performance of repository queries. When a repository is configured, the system creates a variety of object type indexes. You can also create type indexes for your special needs by using the MAKE_INDEX function.

By default, type tables and indexes are stored in the same tablespace or segment. However, you can create a repository with separate tablespaces or segments for each (refer to *Content Server Installation Guide* for instructions) or you can move the indexes later, using the MOVE_INDEX function. Indexes that you create can be placed in any directory you wish.

If you want to remove a user-defined index, use the DROP_INDEX administration method. It is strongly recommended that you do not remove any of the system-defined indexes.

The MAKE_INDEX, MOVE_INDEX, and DROP_INDEX administration methods are described in detail in [Chapter 3, Administration Methods](#), in the *Content Server DQL Reference Manual*. Each can be executed through Documentum Administrator, the Apply method, or the DQL EXECUTE statement.

Alternate locations for object-type tables on Oracle and DB2

If you use Oracle or DB2, to improve performance and increase the throughput of the system, you may want to control where repository information is stored. For example, you can store frequently used data on different disks than less-frequently used data. Defining database parameters to store data in different tablespaces also partitions data into smaller, more manageable pieces.

When a repository is created, the system automatically creates object-type tables and indexes in the underlying RDBMS. (The object-type tables and indexes are described in *Content Server Fundamentals*.)

By default, Content Server creates all object-type tables and indexes in the same tablespace. The size and number of the extents allotted for each table are determined by default configuration parameters. You can edit the server.ini file to change the default database configuration parameters when the repository is created, before you start the server.

For complete instructions, refer to the Appendix entitled Defining Oracle or DB2 Database Parameters for Repository Tables, in the *Content Server Installation Guide*.

Dumping and loading a repository

You must have Superuser privileges to perform a dump or load operation. You use dump and load operations to:

- Move a repository from one location to another
- Duplicate a repository

Use dump and load operations if the duplicated repository must have a name or repository ID different from the source repository. If the duplicated repository is for testing purposes and can have the same name and ID as the source repository, you can use the instructions for creating a repository copy in the *Content Server Installation Guide*. (Note that a repository copied using the procedure in the *Content Server Installation Guide* cannot be moved into a server installation that already contains repositories because the aek.key file used by the new copy is not identical to the aek.key file used in the target server installation.)

- Duplicate or move some portion of a repository

A dump operation creates a binary file of objects dumped from a repository. If a dumped object has associated content files, the content files are either referenced by full path or included directly in the dump file. The load operation loads the objects and content files into another repository.

Code page compatibility issues

Dump files are created using the session code page. For example, if the session in which the dump file was created was using UTF-8, the dump file is a UTF-8 dump file. The repository into which you load a dump file must be using the same code page as that in use when the dump file was created.

Supporting object types

Dump and load operations are supported by four object types:

- Dump Record (dm_dump_record)

A dump record object contains information about a specific dump execution. It has an attribute that contains the name of the file containing the dumped information and attributes whose values tell Content Server which objects to copy out into the specified file.

- Dump Object Record (dmi_dump_object_record)

A dump object record object contains information about one specific object that is copied out to the dump file. Dump object record objects are used internally.

- Load Record (dm_load_record)

A load record object contains information about a specific load operation. Its attributes are used by Content Server to manage the loading process. It also has two attributes that contain the starting and ending times of the load operation.

- Load Object Record (dmi_load_object_record)

A load object record object contains information about one specific object that is loaded from the dump file into a repository. Load object record objects are used internally.

For details about the attributes of these object types, refer to [Chapter 2, Object Reference](#), in the *EMC Documentum Object Reference Manual*.

[Dumping a repository, page 48](#), describes how to set up and use the dump operation. [Loading a repository, page 57](#), describes how to set up and use load operations.

For information about moving the dump file, refer to [Moving the dump file, page 57](#).

For information about trace messages related to dump and load procedures, refer to [Generating dump and load trace messages, page 62](#).

Dumping a repository

We recommend that you always run `dmclean` before dumping a repository. This avoids dumping unwanted objects.

To dump a repository, create and save a dump record object. The act of saving the object starts the actual dump operation. The attributes that you set in the dump record object determine:

- What set of objects are dumped

You can dump all or part of a repository. Refer to [Dumping an entire repository, page 49](#), or to [Dumping specific objects, page 49](#), for instructions.

- Whether content is included directly or referenced in the dump file, and if directly included, whether it is compressed

By default, if a dumped object has associated content, the operation places a reference to the content in the dump file. This is described in [Dumping without content, page 52](#). You can set up the dump operation to include the actual content file in the dump file. [Including content, page 53](#), contains information about including content.

Note: If you are dumping a repository with an encrypted storage area, you must include the content files in the dump file. (The content copied into the dump in clear text. It is not encrypted.)

- The cache size used for the operation

Refer to [Setting the cache size, page 53](#), for information about defining the cache size.

- Whether the operation is restartable

Refer to [Using non-restartable dump, page 54](#), for information about this option.

Dumping objects under retention

Note: The information in this section only applies to dump and loads performed using the dump and load methods in scripts or on the command line. The information does not apply to dump and loads that are used to execute object replication jobs.

If a dumped SysObject is associated with a retainer, the dump operation also dumps the retainer. (Retainers record retention policy definitions).

If you dump a retainer object directly, the object identified in the retainer's `retainer_root_id` attribute is also dumped. That object may be a single SysObject or a container such as a folder. If it is a container, the objects in that container are not dumped, only the container itself is dumped.

Dumping an entire repository

To dump the contents of an entire repository, set the `dump_operation` attribute of the dump record object to `full_dochbase_dump`.

By default, a full repository dump includes content files by reference, not directly. To include them directly, refer to [Including content, page 53](#), for instructions.

A full repository dump does not include any DocApps installed in the repository. After you load the dump file into the new repository, you must reinstall the DocApps.

If you set `dump_operation` to `full_dochbase_dump`, Content Server ignores:

- The `restartable` argument in `dump_parameter`
A full repository dump is always restartable.
- The `cache_size` argument in `dump_parameter`
Refer to [Setting the cache size, page 53](#), for information about the cache.
- The `type`, `predicate`, and `predicate2` attributes

Dumping specific objects

To dump only specific objects in a repository, set the `type`, `predicate`, and `predicate2` repeating attributes of the dump record object. The `type` attribute identifies the type of object you want to dump and the `predicate` and `predicate2` attributes define a qualification that determines which objects of that type are dumped. (Refer to [Dump Record, page 226](#), in the *EMC Documentum Object Reference Manual* for a full description of the attributes of a dump record object.)

However, when you dump an object, the server includes any objects referenced by the dumped object. This process is recursive, so the resulting dump file can contain many more objects than the object specified in the `type`, `predicate`, and `predicate2` repeating attributes of the dump record object.

When dumping a type which has a null supertype, the server also dump all the objects whose `r_object_ids` are listed in the ID field of the type.

The ACL associated with a dumped object is also dumped.

Setting the type attribute

The `type` attribute is a repeating attribute. The object type specified at each index position is associated with the WHERE clause qualification defined in the `predicate` at the corresponding position.

The dump operation dumps objects of the specified type and any of its subtypes that meet the qualification specified in the predicate. Consequently, it is not necessary to specify each type by name in the type attribute. For example, if you specify the SysObject type, then Content Server dumps objects of any SysObject or SysObject subtype that meets the qualification.

Use the following guidelines when specifying object types and predicates:

- The object type must be identified using its internal name, such as dm_document or dmr_containment.

Object type definitions are only dumped if objects of that type are dumped or if objects that are a subtype of the type are dumped.

This means that if a subtype of a specified type has no objects in the repository or if no objects of the subtype are dumped, the dump process does not dump the subtype's definition. For example, suppose you have a subtype of documents called proposal, but there are no objects of that type in the repository yet. If you dump the repository and specify dm_document as a type to dump, the type definition of the proposal subtype is not dumped.

This behavior is important to remember if you have user-defined subtypes in the repository and want to ensure that their definitions are loaded into the target repository.

- To dump subtype definitions for types that have no objects instances in the repository or whose objects are not dumped, you must explicitly specify the subtype in the dump script.
- If you have created user-defined types that have no supertype, be sure to explicitly include them in the dump script if you want to dump objects of those types. For example, the following commands will include all instances of *your_type_name*:

```
append,c,l,type
your_type_name
append,c,l,predicate
1=1
```

- If you have system or private ACLs that are not currently associated with an object, they are not dumped unless you specify dm_acl as a type in the dump script. For example, including the following lines in a dump script will dump all ACLs in the repository (including orphan ACLs):

```
append,c,l,type
dm_acl
append,c,l,predicate
1=1
```

You may want to specify a qualification in the predicate to exclude orphaned internal ACLs.

- By default, storage area definitions are only included if content associated with the storage is dumped. If you want to dump the definitions of all storage areas, even

though you may not dump content from some, include the storage type (file store, linked, and distributed) explicitly in the dump script.

- When you dump the dm_registered object type, Content Server dumps only the object (dm_registered) that corresponds to the registered table. The underlying RDBMS table is not dumped. Use the dump facilities of the underlying RDBMS to dump the underlying table.

Setting the predicate attributes

You must supply a predicate for each object type you define in the type attribute. If you fail to supply a predicate for a specified type, then no objects of that type are dumped.

To dump all instances of the type, specify a predicate that true for all instances of the type, such as 1=1.

To dump a subset of the instances of the object type, define a WHERE clause qualification in the predicate attributes. The qualification is imposed on the object type specified at the corresponding index level in the type attribute. That is, the qualification defined in predicate[0] is imposed on the type defined in type[0], the qualification defined in predicate[1] is imposed on the type defined in type[1], and so forth.

For example, if the value of type[1] is dm_document and the value of predicate[1] is object_name = 'foo', then only documents or document subtypes that have an object name of foo are dumped. The qualification can be any valid WHERE clause qualification. (Refer to [Select, page 115](#), in the *Content Server DQL Reference Manual* for a description of a valid WHERE clause qualification.)

The predicate attribute accepts a maximum of 255 characters. If the qualification exceeds 255 characters, place the remaining characters in the predicate2 attribute at the corresponding index level. For example, if the qualification defined for type[0] is 300 characters, you put the first 255 characters in predicate[0] and the remaining 45 in predicate2[0]. When the dump is executed, Content Server concatenates predicate[0] and predicate2[0]. The predicate2 attribute accepts a maximum of 255 characters also.

Important Note: If you use the predicate2 attribute at any index position, you must also set the predicate2 attribute at all index positions before the desired position. Content Server does not allow you to skip index positions when setting repeating attributes. For example, if you set predicate2[2] and predicate2[4], you must also set predicate2[0], predicate2[1], and predicate2[3]. It is valid to set the values for these intervening index positions to a single blank.

Content files and dumping

How the dump operation handles content depends on where the content is stored and how the `include_content` parameter is set in the `dump_parameter` argument of the dump object.

By default, if the content is stored in a file store or content-addressed storage area, the content is not included in the dump file. You can set the `include_content` parameter to include such content. If you are dumping a repository that has encrypted file store storage areas, you must include the content in the dump file. Content Server decrypts the content before placing it into the dump file.

[Dumping without content, page 52](#), describes the default behavior and requirements for handling dump files without content. [Including content, page 53](#), describes how to include content and the requirements for dump files with content.

If the content is stored in a blob or turbo storage area, the content is automatically included in the dump file because the content is stored in the repository.

Content stored in external storage cannot be included in a dump file.

Dumping without content

By default, a dump operation on content in filestores or content-addressed storage does not include content. Instead, when an object with content is dumped, the operation places a reference to the content in the dump file. If the content is stored in a file system, the reference is a file system path. If the object is stored in a content-addressed storage system, the reference is the content's address.

When the dump file is loaded into the target repository, any file systems referenced for content must be visible to the server at the target site. For content in content-addressed storage, the `ca` store object at the target site must have an identical definition as the `ca` store object at the source repository and must point to the same storage system used by the source repository.

In the target repository, the storage objects for the newly loaded content must have the same name as the storage objects in the source repository but the file paths for the storage locations must be different.

The owner of the target repository must have Read permission in the content storage areas of the dumped repository when the load operation is executed. The load operation uses the target repository owner's account to read the files in the source repository and copy them into the target repository.

Including content

To include content in a dump file, specify the `include_content` argument as `T` (`TRUE`) in the dump record object's `dump_parameter` attribute. If the argument is true, when Content Server dumps an object with content, the content is copied into the dump file also. The content must be stored in a file store or content-addressed storage area. Content Server cannot copy content from external storage into a dump file.

In the target repository, the storage objects for the newly loaded content must have the same names as those in the source repository, but the actual directory location, or IP address for a CA store, can be different or the same.

Always include content if you are dumping a repository to make a backup copy, to archive a repository, or to move the content or if the repository includes an encrypted storage area.

Content compression

When you include content, you can create a compressed dump file to save space. To compress the content in the dump file, set the `dump_parameter` attribute to `compress_content = T`.

Content Server automatically decompresses a compressed dump file during a load operation.

Setting the cache size

Content Server uses an in-memory cache to store the object IDs of dumped objects. Before dumping an object, Content Server checks the cache to see if the object has already been dumped.

You can improve the performance of a large dump operation by setting a larger cache size. If you do not specify a cache size, the server uses a default size of 1 MB, which can hold up to 43,690 object IDs.

To increase the cache size, set the `cache_size` argument of the `dump_parameter` attribute to a value between 1 and 100. The value is interpreted as megabytes and defines the maximum cache size. The memory used for the cache is allocated dynamically as the number of dumped objects increases.

Content Server ignores the cache setting when doing a full repository dump.

Using non-restartable dump

You can also improve the performance of a dump operation by creating a non-restartable dump. However, if a non-restartable dump operation fails, you will not be able to restart the dump from the failure point. Instead, you must create a new dump record object to start the dump operation from the beginning.

A dump operation can only be non-restartable if it is a partial repository dump. Full repository dump operations are always restartable.

To create a non-restartable dump, set the `dump_parameter` attribute to `restartable=F`.

Using a script to create a dump file

For dump operations that you execute regularly, we recommend that you write a script that creates and saves the dump object and checks for errors after the execution. Using a script avoids recreating the dump object manually each time you want to perform the task.

To use a script:

1. Write a script that creates the dump object, sets its attributes, saves the object, and checks for errors.

If you don't set the `file_name` attribute to a full path, Content Server assumes the path is relative to the root directory of the server. The file's name must be unique within its directory. (This means that after a successful load operation using the dump file, you must move the dump file to archival storage or destroy it, so that you can successfully execute the script later.)

2. Use IAPI to execute the script. Use the following command line syntax:

```
iapi source_db -Uusername -Ppassword < script_filename
```

where `source_db` is the name of the repository that you want to dump, `username` is the user name of the user who is executing the operation, `password` is the user's password, and `script_filename` is the name of the file you created in [Step 1](#).

3. If the dump was successful, destroy the dump object. If the Save on the dump operation did not return OK, the dump was not successful.

Destroying the dump object cleans up the repository and removes the dump object records and state information that are no longer needed.

Sample script for a full repository dump with content included

The following script dumps an entire repository by setting the `dump_operation` attribute of the dump record object to `full_docbase_dump`, includes the content directly in the dump file, and compresses the content:

```
create,c,dm_dump_record
set,c,l,file_name
/tmp/dumpfile.out
set,c,l,dump_operation
full_docbase_dump
append,c,l,dump_parameter
set,c,l,include_content
T
append,c,l,dump_parameter
compress_content=T
save,c,l
getmessage,c #Check for dump errors
```

Sample script for a partial repository dump

Note: There is a template for a sample script in `%DM_HOME%\install\DBA\dump_template.bat` (`$DM_HOME/install/DBA/dump_template.api`).

The script below has the following characteristics:

- It doesn't copy content files into the dump file.
- It only dumps ACLs associated with a dumped object.
- It does not dump subtype definitions if there are no objects of that subtype.
- It does not dump storage area definitions if the dump does not include any content associated with the storage area.
- It does not dump user-defined subtypes that have no supertype.
- It does not dump job objects.
- It is not restartable.

The script assumes that you want to dump all instances of the types, not just some subset, consequently, the predicates are set as `1=1` (you cannot leave them blank). If you want to dump only some subset of objects or want to include all ACLs, type definitions, or storage area definitions, modify the script accordingly.

Here is the script:

```
create,c,dm_dump_record
set,c,l,file_name
dump file name# Supply your own file name.
# This must be a new file
append,c,l,type
dm_sysobject
append,c,l,predicate
1=1
append,c,l,type
```

```
dm_assembly
append,c,l,predicate
l=1
append,c,l,type
dm_format
append,c,l,predicate
l=1
append,c,l,type
dm_user
append,c,l,predicate
l=1
append,c,l,type
dm_group
append,c,l,predicate
l=1
append,c,l,type
dmi_queue_item
append,c,l,predicate
l=1
append,c,l,type
dmi_registry
append,c,l,predicate
l=1
append,c,l,type
dm_relation
append,c,l,predicate
l=1
append,c,l,type
dm_relation_type
append,c,l,predicate
l=1
append,c,l,type
dmr_containment
append,c,l,predicate
l=1
append,c,l,type
dmr_content
append,c,l,predicate
l=1
append,c,l,dump_parameter
cache_size=60 #set cache size
append,c,l,dump_parameter
restartable=F #non-restartable dump
append,c,l,predicate
l=1
save,c,l
getmessage,c
```

Notes:

- In the append command line, the l is the lowercase letter L.
- If you don't set the file_name attribute to a full path, Content Server assumes the path is relative to the root directory of the server. The file's name must be unique within its directory. (This means that after a successful load operation using the dump file, you must move the dump file to archival storage or destroy it, so that you can successfully execute the script later.)

- If you want to dump user-defined types that have no supertype, add Append methods for each to the script:

```
append, c, l, type
your_type_name
append, c, l, predicate
l=1
```

If the server crashes during a dump operation

If Content Server crashes during a dump operation, there are two alternatives. You can:

- Destroy the dump file (the target file named in the script) if it exists and then re-execute the script.

If the specified file already exists when you try to save a new dump record object, the save will fail. Re-executing the script creates a new dump record object.

- If the dump operation is restartable, fetch the existing dump object from the source repository and save it again. (Saving the object starts the dump operation.) Content Server will begin where it left off when the crash occurred.

Moving the dump file

The dump file is a binary file. If you move a dump file from one machine to another electronically, be sure to use a binary transfer protocol.

If your operating system is configured to allow files larger than 2 GB, the dump file can exceed 2 GB in size. If you create a dump file larger than 2 GB, you cannot load it on a machine that does not support large file sizes or large file systems.

Loading a repository

Loading a repository puts the objects stored in a dump file into the repository.

If the dump file does not include the actual content files associated with the objects you are loading, the operation reads the content from the storage areas of the dumped repository. This means that the owner of the repository that you are loading must have Read privileges at the operating system level for the storage areas in the source repository.

The load operation generates a `dmi_queue_item` for the `dm_save` event for each object of type `SysObject` or a subtype that is loaded into the target repository. The event is queued to the `dm_fulltext_index_user` user account. This ensures that the objects are added to

the target repository's index. You can turn off this behavior. For instructions, refer to [Turning off save event generation during load operations, page 59](#).

Loading a repository is accomplished by creating and saving a load record object. The act of saving the object starts the operation.

Note: The load operation performs periodic commits to the repository. Consequently, you cannot load a repository if you are in an explicit transaction. The Content Server won't allow you to save a load record object if you are in an explicit transaction. Similarly, you cannot perform a revert or destroy operation on a load record object if you are in an explicit transaction.

Refreshing repository objects from a dump file

Generally, when you load objects into a repository, the operation does not overwrite any existing objects in the repository. However, in two situations overwriting an existing object is the desired behavior. These situations are:

- When you are replicating content between distributed storage areas
- When you are restoring archived content

In both situations, the content object that you are loading into the repository may already exist. To accommodate these instances, the load record object has a relocate attribute. The relocate attribute is a Boolean attribute that controls whether the load operation assigns new object IDs to the objects it is loading.

The type and predicate attributes are for internal use and cannot be used to load documents of a certain type.

Loading job objects

If you dump and load job objects, the load operation automatically sets the job to inactive in the new repository. This ensures that the job is not unintentionally started before the load process is finished and it allows you the opportunity to modify the job object if needed. For example, you may want to adjust the scheduling to coordinate with other jobs in the new repository.

The load operation sets jobs to inactive (`is_inactive=TRUE`) when it loads the jobs, and sets the jobs' `run_now` attribute to `FALSE`.

If the load operation finds an existing job in the target repository that has the same name as a job it is trying to load, it does not load the job from the dump file.

Loading registered tables

When you load a registered table, the table permits defined for that table are carried over to the target repository.

Turning off save event generation during load operations

During a load operation, every object of type SysObject or SysObject subtype loaded into the target repository generates a save event. The event is queued to the dm_fulltext_index_user. This behavior ensures that the object is added to the target repository's index.

The behavior is controlled by the load parameter called generate_event. The parameter is T by default. If you do not want the load operation to queue save events to the dm_fulltext_index_user, set the parameter to F for the operation. The parameter is set in the load_parameter attribute as:

```
generate_event=F
```

Loading a new repository

New repositories are not empty. They contain a variety of cabinets and folders created by the installation process, such as:

- A user object for the repository owner
- A cabinet for the repository owner
- The docu group
- The System cabinet, which contains a number of subfolders
- The Temp cabinet

When you load a dump file into a new repository, these objects are not replaced by their counterparts in the dump file because they already exist in the new repository.

However, if you have changed any of these objects in the source repository (the source of the dump file), the changes are lost because these objects are not loaded. For example, if you have added any users to the docu group or if you have altered permissions on the System cabinet, those changes are lost.

To ensure that any changes you have made are not lost, fetch from the source repository any of the system objects that you have altered and then use the Dump method to get a record of the changes. For example, if the repository owner's cabinet was modified, use the following command sequence to obtain a listing of its attribute values:

```
fetch,c,cabinet_id
```

dump, c, l

After the load operation, you can fetch and dump the objects from the new repository, compare the new dump results with the previous dump results, and make any necessary changes.

The preLoad utility

Documentum provides a utility that you can run on a dump file to tell you what objects that you must create in the new repository before you load the dump file. The utility can also create a DQL script that you can edit and then run to create the needed objects. The syntax for the preload utility is:

```
preload repository [-Username] -Ppassword -dump_file filename  
  [-script_file name]
```

The *repository* is the name of the repository into which you are loading the dump file. *filename* is the name of the dump file, and *name* defines a name for the output DQL script. If you do not include a username, the current user is assumed.

Note: This utility does not report all storage areas in the source repository, but only those that have been copied into the dump file.

The load procedure for new repositories

Use the following procedure to load a dump file into a new repository.

Note: You cannot perform this procedure in an explicit transaction because the load operation performs periodic commits to the repository. Content Server does not allow you to save the load record object to start the load operation if you are in an explicit transaction.

To load a dump file into a new repository:

1. Create the new repository.

Notes:

- If the new repository shares any directories with the source repository, you must assign the new repository an ID that differs from that of the source repository.
 - If the old and new repositories have different owners, make sure that the new repository's owner has Read privileges in the storage areas used by the old repository if the old repository was not dumped with the `include_content` attribute set to TRUE.
2. Create the necessary storage objects and associated location objects in your new repository.

Each storage object in your source repository must have a storage object with the same name in the new repository. The filestore objects in the new repository must reference location objects that point to actual directories that differ from those referenced by the location objects in the source repository.

For example, suppose you have a file store object with the name `storage_1` in your source repository that points to the location object named `enr_store`, which references the `d:\documentum\data\enr (/u04/home/installation_owner/data/enr)` directory. In the new repository, you must create a file store object with the name `storage_1` that references a location object that points to a different directory.

Note: The location objects can be named with different names or they can have the same name. Either option is acceptable.

3. If your storage areas in the source repository had associated full-text indexes, create corresponding fulltext index objects and their location objects in the new repository. Note that these have the same naming requirements as the new storage objects described in [Step 2](#).

4. Create and save the following script:

```
create,c,dm_load_record
set,c,l,file_name
full_path_of_dump_file
save,c,l
getmessage,c
```

5. Log in as the owner of the installation and use IAPI to execute the script.

When you start IAPI, connect to the new repository as a user who has Sysadmin privileges in the repository. (Refer to [Appendix B, IAPI and IDQL](#), for information about using IAPI.)

6. After the load is successfully completed, you can destroy the load object:

```
destroy,c,load_object_id
```

Notes:

- Destroying the load object cleans up the load object record objects that are generated by the loading process and old state information.
- If you created the dump file using a script, after you successfully load the file, move the dump file to archival storage or destroy it. You will not be able to successfully execute the script again if you leave the dump file in the location where the script created it. Content Server will not overwrite an existing dump file with another dump file of the same name.
- If Content Server crashes during a load, you can fetch the Load Object and save it again, to restart the process. Content Server will begin where it left off when the crash occurred.

DocApps

DocApps are not dumped when you dump a repository. Consequently, after you load a new repository, install and run the DocApp installer to reinstall the DocApps in the newly loaded repository.

Generating dump and load trace messages

You can activate tracing during dump and load operations to generate trace messages in the Content Server session log.

To activate tracing, use a Trace method. There are two trace levels that you can turn on for normal dump and load operations using this method:

- Level 8 provides trace information for each object that is dumped or loaded.
- Level 10 provides the level 8 information and also performs a commit after each object is loaded.

In addition to the two levels for normal dump and load operations, trace level 11 provides information for debugging if you are having a problem with a load operation. Generally, load operations return status messages to the server log file after every 100 objects are loaded. If you are having a problem with a load, you can set the trace level to 11 and the operation will return status messages for each object loaded. However, setting the trace level to 11 severely impacts the performance of the load operation. In addition to the extra status messages for the load operation, you also receive all the other messages specified by the lower levels.

The trace information also includes:

- Whether Content Server fails to dump or load an object
- The query used to search for matching objects for a dump or load operation
- The current progress and status of a dump or load operation

For more information about the Trace API method, refer to [Trace, page 456](#), in the *Content Server API Reference Manual*.

Creating location and mount point objects

The directories that a Content Server accesses are defined for the server by location objects. For example, Content Server looks at the location object named `verity` to find out where the Verity TDK full-text indexing executables are located. A location object can represent the location of a file or a directory.

A mount point object represents a directory that is mounted by a client. It is a useful way to aggregate multiple locations that must be mounted.

Note: A UNIX client can only mount a Windows Content Server disk if the Windows machine has a NFS package installed that exports the disk.

Both locations and mount point objects reside in the repository in the File System folder in the System cabinet.

Location objects

You can create a location object using Documentum Administrator, a DQL CREATE...OBJECT statement, or an API Create method. (For help with the procedure using Documentum Administrator, refer to the Documentum Administrator online help.) Using DQL lets you create the object and set its attributes in one step. If you use the API, you must use separate method statements to create the object, set its attributes, and then save the new object.

For example, the following DQL statement creates a location object, sets its attributes, and saves the object when the statement completes successfully. The location object is named `storage_5` and references a directory on a Windows host:

```
CREATE "dm_location" OBJECT
SET "object_name" = 'storage_5',
SET "file_system_path" = 'd:\documentum\data\storage_5',
SET "path_type" = 'file',
SET "security" = 'private'
```

Here is the same example illustrated for a UNIX host:

```
CREATE "dm_location" OBJECT
SET "object_name" = 'storage_5',
SET "file_system_path" = 'u16/home/installation_owner/data/storage_5',
SET "path_type" = 'file',
SET "security" = 'private'
```

If you used IAPI to create the location, the operation would look like this:

```
API>create,s0,dm_location
. . .
3a000000216753cd2

API>set,c,last,object_name
SET>storage_5
. . .
OK

API>set,c,last,file_system_path
SET>d:\documentum\data\storage_5
. . .
OK

API>set,c,last,path_type
SET>directory
```

```

. . .
OK
API>set,c,last,security
SET>private
. . .
OK
API>save,c,last

```

Saving the location object automatically creates only the final leaf of the directory path referenced by the location. For example, if you issue the statement above, the system only creates the storage_5 directory. If the containing directory structure d:\documentum\data (u16/home/installation_owner/data) did not already exist, the operation would fail.

When you create a location object, you must set the following attributes:

- object_name
- file_system_path

The value in file_system_path must consist of ASCII characters.

- path_type

You may also wish to set the optional security_type attribute. If the location object is associated with a storage area, the setting in the attribute is applied to files stored in the associated storage area. There are three possible values for the security_type attribute.

[Table 2-3, page 64](#), lists the values and the file permissions associated with each.

Table 2-3. security_type attribute settings for location objects

Security Setting	Owner	Group	World
public	Read and write	Read	Read
public_open	Read and write	Read and write	Read and write
private	Read and write	None	None

The default setting for security_type is private.

The remaining attributes are optional and the server will provide default values if they are not set. (Refer to [Location, page 294](#), in the *EMC Documentum Object Reference Manual* for a complete listing of the type's attributes.)

If the location represents a directory or file that users will be sharing through disk mounting, consider placing it under one upper level directory that can be represented by a mount point object. If all shared files and directories are put under one mount point, system administration tasks are simplified and system resources are conserved.

Mount point objects

Mount point objects represent shared (mounted) directories. You can create mount point objects using Documentum Administrator, the DQL CREATE...OBJECT statement, or the API. You must have Sysadmin or Superuser user privileges to create a mount point object. Using Documentum Administrator or DQL allows you to create the mount point object, set its attributes and save it in one operation. Using the API requires several method executions.

When you create a mount point, you must set the following attributes:

- object_name
- file_system_path
- host_name

Additionally, the three attributes that record alias values for the shared (mounted) disk are typically set. (Refer to [Mount Point](#), page 316, in the *EMC Documentum Object Reference Manual* for a complete list of the mount point object's attributes.)

Note: A UNIX client can only mount a Windows Content Server disk if the Windows machine has a NFS package installed that exports the disk.

Platform aliases

When you create a shared disk or mount a disk, you typically define an alias for that shared (mounted) disk for use by clients. A mount point object has three attributes to record the aliases for the directory represented by the mount point object. Each attribute represents the alias for a particular client platform.

If you use them:

- Set `unix_preferred_alias` to the directory name you used to mount the directory.

For example, assume you mounted the /u25 disk using the following command:

```
mount remote_computer:/u25 /r25
```

You would set `unix_preferred_alias` to /r25.

- Set `win_preferred_alias` to the alias drive letter you used to mount the directory.

For example, suppose you mounted the same disk on a windows machine using the following command:

```
net use f: \\remote_computer\share\ name
```

You would set `win_preferred_alias` to f:\.

Alternatively, if you are using UNC naming conventions, set `win_preferred_alias` using the following format:

```
\\machine_name\alias_name\
```

For example,

```
\\cougar\link_01
```

cougar is the name of a Windows host and link_01 is a shared name that points to the file system path of the dm_location object representing the storage area.

- Set mac_preferred_alias to the volume name chosen for the mounted directory.

The mounted directory's volume name is set when the directory is exported through the file-sharing system. It is the name that will appear in the Chooser for that directory.

You can override an alias by adding a DOCBASE_OVERRIDES section to your dmcl.ini file. Refer to [Chapter 5, Clients and Sessions](#), for information.

Format objects

Format objects define file formats. Content Server only recognizes formats for which there is a format object in the repository. When a user imports a document, the format of the document must be a format recognized by Content Server. If the format is not recognized by the server, the user cannot save the document in the repository.

The installation procedure adds a basic set of format objects to the repository. You can add more format objects, delete some of the objects, or change the attributes of any object. (You can query the repository to obtain a list of the format objects in the repository. Refer to [Listing current format objects, page 67](#), for a sample query.)

For a complete list of attributes defined for the format object type, refer to [Format, page 248](#), in the *EMC Documentum Object Reference Manual*.

The topic_transform and topic_format attributes

There are some file formats that the index server cannot index directly. If you want to index content files in those formats, their format object must define an alternate, indexable format for them. Subsequently, when you index a document in one of these non-indexable formats, the server automatically creates a rendition of the file in the alternate format and indexes the rendition.

The topic_transform attribute indicates whether you want the system to transform files into another format before indexing. If you set this attribute to TRUE, you must set the topic_format_name attribute to the name of a directly indexable format. (A format's name is found in the name attribute of its format object.)

The DOS extension attribute

When a user requests a file without specifying a file name for the working copy of the file, the server names the file for the user. If the `dos_extension` attribute of the file's associated format object is set, the server automatically appends that extension to the file name when it copies the file to the client local area or the common area.

Similarly, if the storage object associated with the file's storage area has the `use_extensions` attribute set, the server appends the extension defined in `dos_extension` to the file when it places the file in storage. (Refer to [Providing automatic file extensions](#), page 262, for more information about DOS extensions.)

The `format_class` attribute

The `format_class` attribute of the format object may be set to values that determine which formats are indexed:

- `ftalways`

All renditions in formats whose `format_class` attribute is set to `ftalways` are indexed. For example, if a document has renditions in Microsoft Word and PDF formats and the `format_class` attribute for both formats is set to `ftalways`, both renditions are indexed.

- `ftpREFERRED`

If a document has multiple renditions in indexable formats and one is in a format whose `format_class` attribute is set to `ftpREFERRED`, the rendition in that format is indexed rather than any renditions in other formats, with the exception that any formats whose `format_class` attribute is set to `ftalways` are also indexed. If a document has more than one rendition whose `format_class` attribute is set to `ftpREFERRED`, the first rendition processed for indexing is indexed and the other renditions are not. It is recommended that for any document, only one rendition is in a format whose `format_class` attribute is set to `ftpREFERRED`.

If a document has renditions in four different formats, of which the `format_class` of one is set to `ftpREFERRED` and the `format_class` of the other three is set to `ftalways`, all four renditions are indexed.

Listing current format objects

The formats installed by default when Content Server is installed are listed in `formats.csv`, which is found in `%DM_HOME%\install\tools` (`$DM_HOME/install/tools`). The

formats.csv script is run during server installation to install the default formats. You can examine that file to see the formats installed by default and their settings.

The format objects in your repository may be different from those defined in the formats.csv file if you have customized any formats or added formats. Use the following query to obtain a list of the formats currently defined in a repository:

```
SELECT "name", "description" FROM "dm_format"
```

You can modify the query to return other or all format attribute values if you wish to obtain more information about the format definitions in the repository.

Adding format objects

You can add a format object using Documentum Administrator, DQL, or the API. For instructions on using Documentum Administrator, refer to the Documentum Administrator online help. [Format, page 248](#), in the *EMC Documentum Object Reference* lists the attributes of format objects.

Using DQL

Use the DQL CREATE...OBJECT statement to create a format object. The syntax is:

```
CREATE "dm_format" OBJECT
SET attribute = value {,SET attribute = value}
```

For example, the following statement (taken from the formats.ebs script), creates the format object for the text format:

```
CREATE "dm_format" OBJECT
SET "name" = 'text',
SET "can_index" = true,
SET "mac_creator" = 'ttxt',
SET "mac_type" = 'TEXT',
SET "dos_extension" = 'txt',
SET "description" = 'ASCII text'
```

Using the API

To use the API, you must issue separate methods to create the object, set its attributes, and then save the new format. For example:

```
API>create,c,dm_format
. . .
27000001613c45ab
```

```
API>set,c,27000001613c45ab,name
SET>myformat
. . .
OK
API>set,c,27000001613c45ab,can_index
SET>T
. . .
OK
API>save,c,27000001613c45ab
```

Rich media formats

If the format you are adding is a rich media format (video, audio, and so forth), there are four attributes that you may want to set, depending on the format. These attributes must be set using DQL or the API. You cannot set them through Documentum Administrator. The attributes are:

- `richmedia_enabled`

Set this to TRUE if you want to generate thumbnail files, automatic renditions, and metadata for content files in the new format. You must have Documentum Media Transformation Services installed.
- `asset_class`

Set this to enable applications to identify the kind of content represented by the format. Set the attribute to a string value that identifies the content.
- `default_storage`

Set this if you want to store all content files in this format in a storage area that differs from the default storage area for the object type of the object containing the file. Set the attribute to the object ID of the desired storage area.
- `filename_modifier`

Set this if you want WebCache™ to append a modifier to renditions when exporting multiple renditions of the file. Set the attribute to the modifier you want to append.

Modifying formats

You can use Documentum Administrator, DQL, or the API to modify a format object.

Using DQL

Use the DQL UPDATE...OBJECT statement to change a format object. The syntax is:

```
UPDATE type_name [(ALL)] [correlation_variable] OBJECT[S] update_list
[WHERE qualification]
```

where *update_list* is:

```
set attribute_name = value
set attribute_name[[index]] = value
append [n]attribute_name = value
insert attribute_name[[index]] = value
remove attribute_name[[index]]
truncate attribute_name
[un]link 'folder path'
move [to] 'folder path'
```

For example, here is an excerpt of an IDQL session that sets the `dos_extenstion` attribute for a user-defined format:

```
1>update dm_format object
2>set dos_extension = 'txt'
3>where name = 'myformat'
4>go
```

For a complete description of this statement and its use, refer to [Update...Object](#), page 158, in the *Content Server DQL Reference Manual*.

Using the API

If you use the API to modify a format object, you must checkout or fetch the format object before modifying the attributes.

To change a format using the API:

1. Use the Retrieve method to obtain the format object's object ID:

```
API>retrieve,c,dm_format where name = 'format name'
```

2. Check out (or fetch) the format object.

For example:

```
API>checkout,c,returned format id
```

3. Modify the attributes, using the Set method.

```
API>set,c,returned format id,attribute_name
SET>new_value
. . .
OK
```

4. Check in (or save) the format object.

For example:

```
API>checkin,c,returned format id
```

Deleting formats

You cannot delete a format if the repository contains content files in that format.

To delete a format, you can use Documentum Administrator, DQL, or the API.

Using DQL

Use the DQL DELETE...OBJECT statement to delete a format object. The syntax is:

```
DELETE type_name [(ALL)] [correlation_variable] OBJECT[S]  
[WHERE qualification]
```

The qualification in the WHERE clause identifies the format to delete. For example, here is an excerpt of an IDQL session that deletes the format named myformat:

```
1>delete dm_format object  
2>where name = 'myformat'  
4>go
```

For a complete description of this statement and its use, refer to [Delete...Object, page 89](#), in the *Content Server DQL Reference Manual*.

Using the API

The Destroy method requires the format object's object ID as an argument. To obtain that ID, use a Retrieve method.

To use API methods to delete a format object:

1. Use the Retrieve method to obtain the format object's object ID.

```
API>retrieve,c,dm_format where name = 'format name'
```

2. Use the Destroy method to delete the format.

```
API>destroy,c,returned_format_id
```

Alias sets

Alias sets are objects that define one or more aliases and their corresponding values. Aliases are place holders for values in:

- The `r_accessor_name` attribute in template ACLs
- The `acl_name`, `acl_domain`, and `owner_name` attributes in SysObjects
- The `performer_name` attribute in `dm_activity` objects (workflow activity definitions)
- The folder path in Link and Unlink methods

(For a complete description of how aliases are used, refer to [Appendix A, Aliases](#), in the *Content Server Fundamentals*.)

Creating an alias set

Any user can create an alias set. You can use Documentum Administrator, DQL, or the API to create an alias set.

In DQL, use a CREATE...OBJECT statement. In the API, use a Create method to create the object, Set methods to set its attributes, and a Save method to save it to the repository. [Alias Set, page 66](#), in the *EMC Documentum Object Reference* lists the attributes of an alias set.

If you are creating an alias set to use in a template, it is not necessary to provide actual values for the aliases defined in the alias set. If the alias values are undefined, the values are determined when the template is used. For example, if the template is a workflow template, the person starting the workflow is prompted for the alias values when the workflow is started.

Modifying or deleting an alias set

Content Server enforces the following constraints if you use the API to change or delete an alias set:

- To change the owner of an alias set, you must be either the owner of the alias set or a Superuser.
- To change other attributes or to delete an alias set, you must be the owner of the alias set or a user with Sysadmin or Superuser privileges.

If you are using Documentum Administrator to change or delete the alias set, then you must be either the alias set's owner or a Superuser.

Working with object types

Because every business has requirements that are particular to its environment, Documentum allows you to create object types and modify existing object types (with some constraints).

Creating a user-defined type

You can use Documentum Administrator or DQL to create a new object type. For instructions on using Documentum Administrator to create a type, refer to the Documentum Administrator online help.

To create a new type, you must have Superuser, Sysadmin, or Create Type user privileges. You can create a subtype of any of the following object types:

- dm_sysobject and its subtypes
- dm_user and its subtypes
- dm_relation
- dm_state_extension
- dm_state_type
- dm_email_message
- user-defined types

If you have Superuser privileges, you can create a subtype with no supertype.

Using DQL

Use the DQL CREATE...TYPE statement to create a new object type. The statement's syntax and usage information are found in [Create Type, page 71](#), in the *Content Server DQL Reference Manual*.

Be sure to set the acl_domain and acl_name attributes to define the default ACL for the object type if the new type is subtyped from dm_SysObject, a SysObject subtype, dm_user, or a user subtype.. The ACL is not used to control access to the type. Instead, it can be used to assign default permissions for new objects of the type.

The acl_name attribute is set to the object_name of the ACL you are assigning to the object type. You can specify any ACL that you own or that is owned by the repository owner (or the owner's alias, dm_dbo if your RDBMS is Oracle or Sybase). If you identify an ACL owned by the repository owner or dm_dbo in acl_name, you must specify acl_domain. If you specify only acl_name, the server searches only your privately owned ACLs for the ACL.

The `acl_domain` attribute is set to the name of the user who owns the ACL. Valid user names are your name or the repository owner's name or alias (`dm_dbo`).

Unless you are a superuser, you must identify the new type's supertype. If you are a superuser and want to create the type without a supertype, specify `NULL` as the supertype.

To illustrate the statement's usage, here are two examples:

```
CREATE "mytype" TYPE
("name" char(64), "address" char(255), "dependents"
char(32) repeating)
WITH SUPERTYPE NULL

CREATE "report_doc" TYPE
("monthly_total" integer repeating, "month_name"
char(12) repeating)
WITH SUPERTYPE "dm_document"
```

Modifying an object type

Use DQL or the API to modify a system-defined object type. To modify a user-defined object type, use Documentum Administrator.

You can change the default group, default ACL, or default storage area for any object type. However, what other changes you can make depends on whether the object type is a user-defined type or a system-defined type. For a summary of the kinds of changes that you can make, refer to [Table 2–3, page 47](#), of the *Content Server DQL Reference Manual*.

Any changes you make to a type are cascaded to all objects of that type, to its subtypes, and to all objects of any of its subtypes.

Note on deleting attributes

Attributes are stored as columns in a table representing the type in the underlying RDBMS. However, not all RDBMSs allow you to drop columns from a table. Consequently, if you delete an attribute, the corresponding column in the table representing the type may not actually be removed. In such cases, if you later try to add an attribute to the type that has the same name as the deleted attribute, you will receive an error message.

Using DQL

Use the DQL ALTER TYPE statement to modify a user-defined type's definition or set the default group, ACL, or storage area for a type. The statement has four forms, depending on what kind of alteration you want to make.

Changing the default permissions or default storage area

To change a default ACL or storage area, use the SET clause in the ALTER TYPE statement.

The syntax is:

```
ALTER TYPE type_name SET set_clause
```

where *set_clause* is one of the following:

```
DEFAULT STORAGE new storage area
DEFAULT ACL acl_name [IN acl_domain]
```

For example, the following statement changes the default storage area for the report_doc type to storage_12:

```
ALTER TYPE "report_doc" SET DEFAULT STORAGE "storage_12"
```

If the value for *acl_name* or *acl_domain* includes a space or another character that requires you to enclose the string in single quotes, enclose both strings in single quotes. For example:

```
ALTER TYPE "report_doc" SET DEFAULT ACL "design_state" IN "howardj"
```

Adding an attribute

You can add attributes only to custom object types. You cannot add attributes to object types whose names begin with dm. . To add an attribute, use the following syntax:

```
ALTER TYPE type_name ADD attribute_def {,attribute_def}
```

The *type_name* argument must identify a user-defined type.

The *attribute_def* argument defines the new attribute's name, datatype, length if it is a string datatype, and whether it is a repeating attribute. The format for this information is:

```
attribute_name datatype [REPEATING]
```

For example, the following statement adds a character string attribute to the report_doc type:

```
ALTER TYPE report_doc ADD department string (16)
```

If you want department to be a repeating attribute, the statement looks like this:

```
ALTER TYPE "report_doc" ADD "department" string (16) REPEATING
```

The definition can also include data dictionary information for the attribute. For information about the syntax for defining data dictionary information, refer to [Alter Type, page 45](#) in the *Content Server DQL Reference Manual*.

Deleting an attribute

You can delete attributes only from user-defined object types. Use the following ALTER TYPE syntax:

```
ALTER TYPE type_name DROP attribute_name
```

The *type_name* argument must identify a user-defined type.

For example, the following statement deletes the department attribute from the report_doc type:

```
ALTER TYPE "report_doc" DROP "department"
```

Lengthening a string attribute

You can lengthen string attributes only in user-defined object types. Use the following ALTER TYPE syntax:

```
ALTER TYPE type_name MODIFY attribute_def {,attribute_def}
```

The *type_name* argument must identify a user-defined type.

The *attribute_def* argument identifies both the attribute to change and its new length. For example, suppose you want to lengthen the department attribute for the report_doc type to 24 characters. Here is the statement:

```
ALTER TYPE "report_doc" MODIFY "department" char(24)
```

If department is a repeating attribute, the statement looks like this:

```
ALTER TYPE "report_doc"  
MODIFY "department" char(24) REPEATING
```

Deleting a type

You can only remove a type from the repository if:

- The type is user-defined type.
You cannot remove system-defined types from the repository.
- You are the owner of the type or a user with Superuser user privileges.

- The type has no subtypes.
- There are no existing objects of that type in the repository.

You can use either Documentum Administrator or DQL to remove a type from the repository. For instructions on using Documentum Administrator, refer to the Documentum Administrator online help.

To use DQL, use the DROP TYPE statement. The syntax is:

```
DROP TYPE type_name
```

For example, the following statement removes the report_doc type:

```
DROP TYPE "report_doc"
```

Cleaning up repositories

Every site should have a schedule for repository maintenance that includes regular repository clean up. Cleaning up a repository involves removing:

- Orphaned content files

When users delete a document, or any object that has a content file associated with it, the system deletes the object and marks the content as an orphan. The system does not delete the actual content file. This must be done using the dmclean utility.

- Unwanted document versions and renditions

Depending on your business rules, you may want to remove older versions of a document. You will also want to remove renditions associated with deleted documents or unneeded renditions and annotations for current documents.

- Orphaned annotations and internal ACLs

An annotation (dm_note object) is orphaned when it is detached from all documents or other objects to which it was attached.

An internal ACL (dm_acl object) is orphaned when it is no longer referenced by any object. (Internal ACLs are ACLs that are created by the server.)

- Aborted workflows

A workflow that has been stopped by the execution of an Abort method is an aborted workflow.

- Old log files

Cleaning up a repository regularly helps ensure that there is little or no wasted space in your installation.

Use the following procedure to clean up a repository:

1. Back up the repository completely.

2. Delete any unwanted versions of documents.

Based on your business rules, you may want to delete only versions created before a certain date or by a certain author. Or, you may want to delete all but the CURRENT version from one or more version trees. What you want to delete will determine which method you use.

- To delete selected versions of documents, use the DELETE...OBJECT statement.

You can identify the documents to delete by their creation date, modification date, or some other criteria that you choose. For example, the following statement deletes all documents that have not been changed since January 1, 1998:

```
DELETE "dm_document" OBJECTS
WHERE "r_modify_date" < DATE('01/01/1998')
```

Or, the next statement deletes all documents that were created before January 1, 1997 and that have the version label outdated:

```
DELETE "dm_document" OBJECTS
WHERE "r_creation_date" < DATE('01/01/1997')
AND ANY "r_version_label" = 'outdated'
```

- To delete versions from a version tree, use the Prune method.

Prune deletes all unwanted versions on a specified tree or branch of a tree. An unwanted version is any version that has no symbolic label and that does not belong to a virtual document. Refer to [Prune, page 326](#), in the *Content Server API Reference* for the usage of the method.

If you specify the root version of the version tree in the Prune method, the method searches the entire tree for versions to prune. If you specify a node on the tree, the method only searches the versions on that branch of the tree.

The *keepSLabel* argument is a flag that tells the server whether or not to keep any version that has a symbolic label. It is TRUE by default. If you set it to FALSE, the server removes versions with symbolic labels.

3. Delete unused renditions.

Renditions are created automatically by the server to fulfill a user's request or explicitly by a user. A rendition is represented in the repository by a content object that points to the rendition's source document and by a content file.

Over time, you may find that you want to remove document renditions that are no longer needed or wanted. To delete a rendition (without deleting its source document), you first update the content object for the rendition to remove its reference to the source document. For example, the following UPDATE...OBJECT statement updates all server- and user-generated renditions created before January 1, 1998. The updates in the statement detaches the affected renditions from their source documents, effectively deleting them from the repository.

```
UPDATE "dmr_content" OBJECTS
SET "parent_count" = 0,
```

```
TRUNCATE "parent_id",
TRUNCATE "page"
WHERE "rendition" != 0 AND "set_time" < DATE('01/01/1998')
```

The content objects for the renditions and their content files are now orphaned and you must run `dmclean` to remove them ([Step 6](#)).

4. Clean up the temp directory by deleting the temporary files in that location.

You can determine where the temp directory is with the following query:

```
SELECT "file_system_path" FROM "dm_location"
WHERE "object_name" = 'temp'
```

5. Delete any unwanted `dmi_queue_item` objects.

Every time an object is placed in a user's inbox, a `dmi_queue_item` object is created. When the object is removed (by any means), the queue item object is not destroyed, but it is marked in the repository as dequeued. Based on your business rules, you may want to remove some of these queue item objects. You can use the `DELETE...OBJECT` statement for this. For example, the following statement removes all queue items objects representing objects that were dequeued before January 1, 1998:

```
DELETE "dmi_queue_item" OBJECTS
WHERE "dequeued_date" < DATE('01/01/1998')
AND "delete_flag"=true
```

6. Run `dmclean` to remove the content files left orphaned when you removed old versions and renditions, orphaned annotations and ACLs, and aborted workflows. You can execute the `Dmclean` administration tool or run the `dmclean` utility manually. Instructions for using the tool are found in [Dmclean](#), page 494. Instructions for running the utility manually are found in [Using dmclean](#), page 270.
7. Delete or archive old server logs, session logs, trace files, and old versions of the product.

Session logs are found in `%DOCUMENTUM%\dba\log\repository_id` (`$DOCUMENTUM/dba/log/repository_id`).

Content Server and connection broker log files are found in `%DOCUMENTUM%\dba\log` (`$DOCUMENTUM/dba/log`). The server log for the current server session is named `repository_name.log`. The log for the current instance of the connection broker is named `docbroker.docbroker_hostname.log`. Older versions of these files have the extension `.save` and the time of their creation appended to their name.

On Windows, you can use the `del` command or the File Manager to remove unwanted session logs, server logs, and connection broker logs. . On UNIX, use the `rm` command.

Maintaining query performance

As users add documents and other objects to the repository, the tables in the RDBMS that store information about those objects will grow. This affects the performance of queries. (This can be particularly noticeable for queries against documents stored in a distributed storage area as the `dmi_replica_record` table grows.) To ensure that queries are using the best possible query plan, be sure to run statistics against your repository regularly.

Documentum provides a system administration tool called Update Statistics that generates statistics for all the repository tables. For information about this tool, refer to [Update Statistics](#), page 530.

Alternatively, you can use run statistics using the command available directly in the RDBMS. For information about that, consult the documentation provided by your RDBMS vendor.

Configuring repository-level package name control

When an application or user issues an `addPackage` or `addAttachment` method to add a package or attachment to an activity or work item in a running workflow, the application or user has the option of providing the names of the package or attachment components as a method argument. By default, when the names are provided, Content Server records the names in the generated package or wf attachment object. For packages, this makes the name available for use in the message string specified in the activity's `task_subject` string. (Attachments are not supported for referencing in task subjects.) However, for security reasons, you may wish to disable the ability to record the names in the package or wf attachment object by turning on package control.

Two attributes control this functionality:

- `wf_package_control_enabled` in the docbase config object
- `package_control` in the workflow definition (a `dm_process` object)

The `wf_package_control_enabled` attribute is set to F (FALSE) by default. This means that Content Server can record component names in package or wf attachment objects if package control is not enabled in the workflow and the names are provided in `addPackage` or `addAttachment` method. When `wf_package_control_enabled` is set to F, the setting of the `package_control` attribute in the workflow definition determines whether the name is actually recorded in the package object.

The `package_control` attribute in the workflow definition (`dm_process` object) is set to 0 by default. This setting means that package control is not enabled, allowing the server

to record package names specified in `addPackage` and `addAttachment` methods in the package or wf attachment objects generated for the workflow's activities or work items.

Table 2-4, page 81, illustrates how the attribute settings interact to determine whether the server can record the names. Note that enabling package control at the repository level (setting `wf_package_control_enabled` to T) means that the server cannot record the names regardless of the workflow setting.

Table 2-4. Interaction of attributes determining package control behavior

package_control (process setting)	wf_package_control_enabled (docbase config setting)	
	T (on)	F (off)
0 (off)	Content Server records blanks in <code>r_component_name</code>	Content Server records object names in <code>r_component_name</code> if specified in <code>Addpackage</code>
1 (on)	Content Server records blanks in <code>r_component_name</code>	Content Server records blanks in <code>r_component_name</code>

To enable package control at the repository level, which means that the server cannot record component names in the package object regardless of the setting in the workflow definition, set the `wf_package_control_enabled` attribute in the docbase config object to T (TRUE).

Configuring a repository printer on Windows

If the repository is running on a Windows host, you can set up a printer to enable users and applications to print documents from the repository using the Print API method.

Printer setup requires the Docbasic script `printnt.ebs`, which is found in the `%DOCUMENTUM%\bin` directory.

To configure a printer:

1. Create a `dm_method` object with a method type set to `dmbasic` and the `method_verb` set to

```
dmbasic -f printnt.ebs -e Entry_Point
```

For detailed information about creating method objects, refer to [Chapter 2, Content Repositories](#).

2. Create an output device object (`dm_outputdevice`) for the printer with the `output_formats` attribute set to all supported formats. The default supported formats are:

- `msw6`
- `msw8`
- `msww`
- `text`
- `crtxt`

You can only print documents which are in the formats listed in the `output_formats` attribute. You can add additional formats.

To add additional formats:

1. Write a new routine, similar to the `Print_Word` routine in `printnt.ebs`, that uses OLE to print the new format.
2. Link in the new routine by adding an `If ... Then` clause to the `Print_File` routine in `print.ebs`.
3. Add the new format to the `output_formats` attribute of the appropriate `dm_outputdevice` object.

Servers

This chapter contains an overview of Content Servers and the standard procedures for working with the servers. The topics in this chapter are:

- [Overview of servers, page 83](#)
- [Internationalization, page 86](#)
- [The dm_start_docbase script \(UNIX\), page 87](#)
- [The server.ini file, page 88](#)
- [Moving the server executable \(UNIX only\), page 106](#)
- [Changing default operating system permits on directories and files \(UNIX only\), page 106](#)
- [Changing a server's configuration, page 107](#)
- [Setting the secure connection mode, page 109](#)
- [Restarting a server, page 109](#)
- [Starting additional servers, page 110](#)
- [Communicating with connection brokers, page 112](#)
- [Shutting down a server, page 118](#)
- [Stopping a session server, page 120](#)
- [Server log files, page 121](#)
- [Server load balancing and failover, page 122](#)
- [Clearing the server common area, page 123](#)
- [Adding additional servlets to the Java method server, page 123](#)
- [Configuring the workflow agent, page 124](#)
- [Recovering automatic activity work items on Content Server failure, page 125](#)

Overview of servers

Servers are processes that provide client access to the repository. They receive queries from clients in the form of API methods or DQL statements and make a call to the

underlying RDBMS or the file directories. Every repository must have at least one active server. If a repository does not have an active server, then users cannot access that repository.

Server threads (Windows)

When a server is started, the resulting server process contains three threads, one of which is called the *main thread*. When a client asks for a repository connection through the server, a session thread is started for that client within the server process. Session threads persist only for the life of the session. As soon as the session terminates, so do they.

The number of session threads that can be started within the server process is configurable.

Parent servers and session servers (UNIX)

When a server is started, the resulting server process is called a parent server. Each time a client asks for a repository connection through a parent server, the parent server spawns another server process to service that client. These spawned server processes are called *session servers* in this manual. They persist only for the life of the session. As soon as the session terminates, so do they.

The number of session servers that can be spawned from a parent server is configurable.

Configuration

A server configuration is defined by the server's `server.ini` file and server config object. Both are created during the installation procedure for the server and called when the server is started.

The `server.ini` file contains information you provide during the installation process, including the repository name and the repository ID. That information allows the server to access the repository and contact the RDBMS server. The `server.ini` file also contains the name of the server's server config object.

The attributes in the server config object give the server its operating parameters and provide it with a map to the files and directories that it will access during the course of its work. For example, the `concurrent_sessions` attribute tells the server how many concurrent users it can accept, and the `verity_location` attribute tells the server where to find the Verity full-text executables.

Multiple servers

The standard, default installation process creates one repository with one server. After you complete the procedure, you can add additional servers for the repository. If you are implementing a configuration that uses a distributed storage area, you will install a server at the site of each distributed component. You may also want to start additional servers for load balancing or to enhance performance if a repository is very active or serving many users, or if its users are spread over a wide area.

Servers, connection brokers, and clients

The connection broker is the intermediary between the client and the server when a client wants a repository connection. If a server is not known to at least one connection broker, no clients can connect to the repository associated with the server.

Each server regularly projects connection information to at least one connection broker. When a client requests a connection to a repository, the connection broker sends the client the connection information for each server associated with the repository. The client can then choose which server to use. (Refer to [Chapter 6, Connection Brokers](#), for more information about connection brokers. Clients and client sessions are the subject of [Chapter 5, Clients and Sessions](#).)

You can configure the server to project information to multiple connection brokers. You can also configure how often the server projects information.

The agent exec process

The agent exec process is the process that oversees the execution of jobs (Jobs are automated methods. For information about jobs, refer to [Chapter 4, Methods and Jobs](#).) An agent exec process is installed with each Content Server.

When you start a server, the agent exec process is automatically started by an invocation of the `agent_exec_method`. The `agent_exec_method` method is created by the `headstart.ebs` script when you configure the repository, and its name is recorded in the `agent_launcher` attribute of the server config object.

The agent exec process runs continuously, polling the repository at specified intervals for jobs to execute. The default polling interval is 60 seconds. The polling interval can be changed. (Refer to [Modifying agent exec behavior](#), page 152, for instructions.)

If there are multiple jobs to execute, the process runs each job, sleeping for 30 seconds between the jobs. The maximum number of jobs that the process can execute in each polling cycle is controlled by the `max_concurrent_jobs` key in the `dmcl.ini` file. By

default, the process runs up to three jobs in each polling cycle. After the jobs are executed and the polling cycle is complete, the process sleeps for the specified interval before polling the repository again.

You can change the agent exec's polling interval or modify the maximum number of jobs run in a polling cycle. However, you cannot change the 30-second sleep period between jobs in a polling cycle. For instructions on changing the configurable defaults, refer to [Modifying agent exec behavior, page 152](#). This section also includes information about turning on tracing for the agent exec process.

ACS servers

When the first repository is installed and configured in an installation on a host machine, the process also installs an ACS server. This server is used by WDK web-based client applications. For information about this server and the configurations that use it, refer to the *EMC Documentum Distributed Configuration Guide*.

Internationalization

When you install Content Server, the procedure determines the locale of the host machine and, based on that locale, sets Content Server's locale to one of the supported locales. The supported locales are:

- English
- French
- German
- Japanese
- Korean
- Spanish
- Italian

The procedure also uses the host machine's locale as the basis for setting some other configuration parameters that define the expected code page for clients and the host machine's operating system. [Table 3-1, page 87](#), lists these parameters and their default settings for each locale.

Table 3-1. Default code page settings by host machine locale

Parameter	Host Machine Locale						
	English	French	Italian	Spanish	German	Japanese	Korean
locale_name	en	fr	it	es	de	ja	ko
(server config attribute)							
default_client_codepage	ISO_8859-1	ISO_8859-1	ISO_8859-1	ISO_8859-1	ISO_8859-1	Shift_JIS	EUC-KR
(server config attribute)							
server_os_codepage	ISO_8859-1	ISO_8859-1	ISO_8859-1	ISO_8859-1	ISO_8859-1	On Windows: Shift_JIS On UNIX: EUC_JP	EUC-KR
(server config attribute)							

The Content Server uses the UTF-8 code page. If the `server_codepage` key is set in the `server.ini` file, it must be set to UTF-8. While Documentum requires Unicode UTF-8 for Content Server's internal code page, the database (RDBMS) and operating system of the server's host machine may use non-Unicode code pages.

On clients, the `Shift_JIS` code page supports the NEC extensions.

You cannot reset the server's `locale_name`, and it is strongly recommended that you do not reset the `default_client_codepage` or `server_os_codepage`. The server uses the value in `default_client_codepage` when communicating with pre-4.2 Documentum client applications. Resetting this parameter or `server_os_codepage` may cause unexpected errors in how the server handles queries and content files.

The `dm_start_docbase` script (UNIX)

Starting a server invokes the `dm_start_repository` script. The script checks to make sure that a log directory is defined for the installation, copies any existing log file for the server to a new location, and then starts the server. The script has an optional argument, `-oclean`, that removes the files in the server common area if you include it in the command line.

To configure the server being started, the script reads the `server.ini` file and the server config object referenced in the `server.ini` file.

The command line that starts the server executable specifies a relative path for the server executable. The script reads the `DM_HOME_CURRENT` environment variable and sets the current directory to the directory specified in that variable before executing the command line. (`DM_HOME_CURRENT` is set to `$DM_HOME/bin.`)

The starting command line contains an argument called `-security`. This argument is set during Content Server's installation to provide an initial security level for the repository. It is read once, the first time you start the first server for the repository. When you run the script again to restart the server or run an altered script to start additional servers, the `-security` argument is ignored.

The `dm_start_repository` script is stored in the `$DOCUMENTUM/dba` directory.

The server.ini file

The `server.ini` file contains configuration information used by the server to define its behavior. The file is stored in `%DOCUMENTUM%\dba\config\repository` (`$DOCUMENTUM/dba/config/repository`) and is called when the server is started.

The format of the file is:

```
[SERVER_STARTUP]
key=value

[DOCBROKER_PROJECTION_TARGET]
key=value

[DOCBROKER_PROJECTION_TARGET_n] #n can be 0-49
key=value

[FUNCTION_SPECIFIC_STORAGE] #Oracle & DB2 only
key=value

[TYPE_SPECIFIC_STORAGE] #Oracle & DB2 only
key=value

[FUNCTION_EXTENT_SIZE] #Oracle only
key=value

[TYPE_EXTENT_SIZE] #Oracle only
key=value
```

Only the `[SERVER_STARTUP]` section is required. The other sections are optional.

Note: To receive a verbose description of the `server.ini` file, type the following command at the operating system prompt:

- On Windows:

```
dmserver_v4 -h
```

- On UNIX:

```
documentum -h
```

If you want to add a comment to the file, use a semi-colon (;) as the comment character.

SERVER_STARTUP section

The keys in the [SERVER_STARTUP] section provide the information the server needs to access the repository and the database. When you install the server, you are prompted for the information to set these keys. The [SERVER_STARTUP] section also contains keys that provide default operating parameters for the server. You are not prompted for these values. Some are default values and some are optional. You can change the defaults or set optional keys during installation or after the installation process is completed.

[Table 3-2, page 89](#), lists the keys in the server startup section. For look-up convenience, the keys are listed in alphabetical order in the table, though they do not appear in that order in an actual server.ini file.

Table 3-2. Server.ini SERVER_STARTUP section keys

Key	Datatype	Comments
acl_update_threshold	integer	None
check_user_interval	integer	The default is 100.
client_session_timeout	integer	Value is interpreted in minutes.
commit_read_operations	Boolean	None
concurrent_sessions	integer	The default is 100.
data_store	string	Used with DB2 only
database_conn	string	Required by Oracle and DB2.
database_name	string	Not required by Sybase and MS SQL Server Not required by Oracle and DB2.
database_owner	string	Required by Sybase and MS SQL Server. None

Key	Datatype	Comments
database_password_file	string	None
distinct_query_results	Boolean	None
docbase_id	integer	None
docbase_name	string	None
enable_workitem_mgmt	Boolean	The default is F (FALSE).
enforce_four_digit_year	Boolean	The default in a new repository is T.
gethostbyaddr	Boolean	None
history_cutoff	integer	None
history_sessions	integer	None
host	string	None
ignore_client_domain	Boolean	Used on Windows platforms only
index_store	string	Used with DB2 only
install_owner	string	unused. The server config setting is used instead.
mail_notification	Boolean	None
max_ftacl_cache_size	integer	Limits the number of elements cached per session to process FTDQL-compliant full-text queries.
max_session_heap_size	integer	None
method_server_enabled	Boolean	The default is T (TRUE).
method_server_threads	integer	The default is 5.
preserve_existing_types	Boolean	None
rdbms_connect_retry_timeout	integer	None
root_secure_validator	string	
saveasnew_retain_source_group	Boolean	Controls which group is set as default group for an object created with a Saveasnew method.
server_codepage	string	UTF-8 is the only allowed value

Key	Datatype	Comments
server_config_name	string	None
server_startup_sleep_time	integer	None
service	string	Service name for the repository
start_index_agents	Boolean	The default is T (TRUE).
thread_lock_timeout	integer	None
ticket_multiplier	integer	Refer to Setting the ticket cache size for Content Server, page 171 for information about this key.
umask	string(4)	This is supported for UNIX platforms only. Refer to Changing default operating system permits on directories and files (UNIX only), page 106 for information about its use.
update_access_date	Boolean	None
upd_last_chg_time_from_db	Boolean	None
use_estimate_search	Boolean	None
user_auth_case	string	None
user_auth_target	string	Used on Windows platforms only
validate_database_user	Boolean	Controls whether Content Server checks for a valid OS account for the database owner's user account.
wait_for_connect_timeout	integer	None

DOCBROKER_PROJECTION_TARGET sections

The [DOCBROKER_PROJECTION_TARGET] and [DOCBROKER_PROJECTION_TARGET_n] sections define the connection brokers to which the server sends its connection information. When you install Content Server, the procedure creates one

[DOCBROKER_PROJECTION_TARGET] section in the server.ini file, which contains access information needed for a server's first broadcast to a connection broker. In that first section, the host key is set to the connection broker name you provide during the installation. The proximity key is set to a default value of 1. When the server is started at the end of the installation procedure, it projects connection information to the connection broker specified in the host key.

Connection broker projection targets are also defined in a set of attributes in the server config object. Using the server config attributes to define additional projection targets, rather than the server.ini, is recommended because it allows you to change a target without restarting the server. If the same projection target is defined in both the server config attributes and in the server.ini file, the values for the target in the server config attributes are used.

The [DOCBROKER_PROJECTION_TARGET] section defines the first projection target. To define additional targets, use [DOCBROKER_PROJECTION_TARGET_n] sections. n can be any integer from 0 to 49. (Refer to [Defining connection broker projection targets, page 112](#), for instructions about defining these sections.)

Table 3–3, page 92, lists the keys for these sections.

Table 3-3. Server.ini DOCBROKER_PROJECTION_TARGET keys

Key	Datatype	Comments
host	string	name of connection broker host
port	integer	port number used by the connection broker
proximity	integer	user-defined value representing distance of server from connection broker

FUNCTION_SPECIFIC_STORAGE and TYPE_SPECIFIC_STORAGE sections

The [FUNCTION_SPECIFIC_STORAGE] and [TYPE_SPECIFIC_STORAGE] sections define which tablespace or device will store the RDBMS tables and indexes for object types. These sections are available only for Oracle and DB2 and must be defined when Content Server is installed. For information about using these sections, refer to *Content Server Installation Guide*.

Table 3–4, page 93, lists the keys for a FUNCTION_SPECIFIC_STORAGE section.

Table 3-4. Server.ini FUNCTION_SPECIFIC_STORAGE keys

Key	Datatype	Comments
database_table_large	string	Name of a tablespace
database_table_small	string	Name of a tablespace
database_index_large	string	Name of a tablespace
database_index_small	string	Name of a tablespace

[Table 3-5, page 93](#), lists the keys for a TYPE_SPECIFIC_STORAGE section.

Table 3-5. Server.ini TYPE_SPECIFIC_STORAGE keys

Key	Datatype	Comments
database_table_ <i>typename</i>	string	The key is set to the name of a tablespace. Replace <i>typename</i> with the name of the object type.
database_index_ <i>typename</i>	string	The key is set to the name of a tablespace. Replace <i>typename</i> with the name of the object type.

FUNCTION_EXTENT_SIZE and TYPE_EXTENT_SIZE sections

the [FUNCTION_EXTENT_SIZE] and [TYPE_EXTENT_SIZE] sections determine how much space is allocated in the RDBMS for the object type tables. They are available only for Oracle and must be defined when the repository is installed. For information about using these sections, refer to *Content Server Installation Guide*.

[Table 3-6, page 93](#), lists the keys for the FUNCTION_EXTENT_SIZE section.

Table 3-6. Server.ini FUNCTION_EXTENT_SIZE keys

Key	Datatype	Comments
database_ini_ext_large	integer	None
database_ini_ext_small	integer	None
database_ini_ext_default	integer	None
database_next_ext_large	integer	None

Key	Datatype	Comments
database_next_ext_small	integer	None
database_next_ext_default	integer	None

Table 3-7, page 94, lists the keys for the TYPE_EXTENT_SIZE section.

Table 3-7. Server.ini TYPE_EXTENT_SIZE keys

Key	Datatype	Comments
database_ini_ext_ <i>typename</i>	integer	Replace <i>typename</i> with the name of the object type.
database_next_ext_ <i>typename</i>	integer	Replace <i>typename</i> with the name of the object type.

Keys you provide at installation

The keys described in this section derive their settings from information you provide during the server installation procedure.

docbase_id

The docbase_id key contains the repository ID. You will find a range of valid values enclosed in the box with your software. Use one of the numbers in the range you have been assigned. If you have other repositories at your site, the number you select must be unique among all the repositories.

docbase_name

The docbase_name key contains the name you choose for your repository. It can be any name that conforms to the naming rules described in [Names, page 31](#), in the *EMC Documentum Object Reference Manual*.

database_owner

The database_owner key contains the RDBMS login name of the repository's owner.

database_conn

The database_conn key contains the database connection string, which is used by the server to connect with the RDBMS server. This value is required by Oracle and DB2.

Oracle database_conn value

The database_conn value is the alias for the Oracle database. The alias is defined in a file called tnsnames.ora. Here is a sample entry in this file:

```
production=
  (DESCRIPTION=
    (ADDRESS=
      (PROTOCOL=TCP)
      (HOST=muskox)
      (PORT=1232)
    )
    (CONNECT_DATA=
      (SID=ORC)
    )
  )
```

The alias is the name specified on the first line. In the example above, the alias is production. Ask your Oracle System Administrator or DBA for the alias for the database that contains your repository's tablespace.

Sybase database_conn value

The database_conn value is the name of the Sybase server.

MS SQLServer database_conn value

The database_conn value is the ODBC Data Source Name assigned to the SQLServer. This name can be found by double-clicking the ODBC Data Sources icon in the Control Panel.

DB2 database_conn value

The database_conn value is the name of the DB2 database on which the repository is running.

database_name

The database_name key identifies your tablespace or database in the RDBMS. This key is required by Sybase and MS SQL Server. Oracle and DB2 do not use this key.

Sybase database_name value

The database_name is the name of the Sybase database corresponding to your repository.

MS SQL Server database_name value

The database_name is the name of the SQL Server database corresponding to your repository.

service

The service key contains the TCP/IP service name for the Content Server. The value is the service name that appears in the server host machine's services file. If a repository has multiple servers, this name must be unique for each server.

You can specify multiple service names, but the services must be running on the same host machine. Separate the service names with commas. (Supplying multiple names is a fail-over mechanism.)

host

The host key specifies an IP address on which the server will listen. Some host machines have multiple network cards. If you want the server to use a particular network card on the host machine, specify the card's IP address in this key before starting the server.

Optional keys

You are not prompted for values for these keys. Some are given default values during the installation procedure and some are not. You can set them during the installation procedure by modifying the server.ini file before the server is started. Some can also be set after the procedure is finished. (Refer to [Modifying the server.ini file, page 107](#), for information about changing a server.ini file.)

acl_update_threshold

The `acl_update_threshold` key, if set, is used when the permission granted to `dm_world` in an ACL is updated. Setting `acl_update_threshold` improves performance of the update.

Changing the `dm_world` permission in an ACL (or adding an entry for `dm_world`) may cause Content Server to set the `r_is_public` attribute for any objects using the ACL. Changes to `r_is_public` may require related changes to associated content objects.

If `acl_update_threshold` is not set, Content Server iterates through and updates the entire `dmr_content` object type table. If the key is set and the affected rows are fewer than the key value, only the affected rows are updated, which improves performance. However, if the number of affected rows is larger than the threshold set, Content Server ignores the key value and updates the entire table.

check_user_interval

The `check_user_interval` key defines the frequency, in seconds, with which Content Server checks the session user's login status for changes.

The default value is 0, meaning that the user's status is checked only at connection time.

commit_read_operations

The `commit_read_operations` key controls whether an explicit commit call is performed after each read-only repository operation is completed. Under the default setting, `TRUE`, a commit is issued after each read-only repository operation, with two exceptions:

- When the operation is part of a multi-statement transaction
 - In this case, the commit must be issued by the caller.
- When a collection is open

In this case, the commit is issued when the last collection is closed.

`FALSE` means that all update operations are committed when they are complete and read operations execute under the assumption that the RDBMS will release locks automatically at the end of the operation.

data_store and index_store

The data_store and index_store keys are used only by installations running on DB2. The data_store key identifies the tablespace in which the repository tables are to be stored. The index_store key identifies the tablespace in which the type index tables are to be stored.

By default, these keys are set to the default tablespace of the user performing the repository configuration. You can change the default during repository configuration if you use the custom configuration option to configure the repository. The behavior when you set these keys is as follows:

- If you set data_store and don't set index_store, the system creates both the object type tables and the index tables in the tablespace defined by data_store.
- If you set index_store and don't set data_store, the system creates the indexes in the tablespace defined by index_store and creates the object type tables in the user's default tablespace.
- If you set both keys, the system creates the object type tables in the tablespace specified in data_store and the indexes in the tablespace specified in index_store.

Note: On DB2, you cannot move indexes after the index tables are created.

If you uninstall a DB2 repository with separate tablespaces for the object type and index tables, the procedure doesn't destroy the tablespaces, nor does it destroy all the repository tables. It destroys only the following repository tables:

- dmi_vstamp
- dmi_object
- dmi_object_type

enable_workitem_mgmt

The enable_workitem_mgmt key controls whether permissions to perform certain workflow actions are enforced. The affected actions are:

- Acquiring a work item
- Delegating a work item
- Halting and resuming a running activity
- Changing a work item's priority

If the key is set to T (TRUE), any user can perform those actions. The key is F (FALSE) by default.

gethostbyaddr

The `gethostbyaddr` key determines whether the server calls the `gethostbyaddr()` function to obtain the host name of the machine on which a client application resides. By default, this key is `TRUE`.

If a large number of your client machines don't have names, set this to `FALSE` to skip the calls to `gethostbyaddr` during connection requests, resulting in better performance for connection requests. (The server uses client host addresses instead of names instead.)

history_sessions and history_cutoff

The `history_sessions` key defines the number of maximum historical sessions (timed-out sessions) the `LIST_SESSIONS` function will return. (`LIST_SESSIONS` is run using the `Apply` method or the `EXECUTE` statement.)

The `history_cutoff` key specifies a cut-off time for historical sessions in minutes. For example, if you set `history_cutoff` to 15, then the server will not return any historical session older than 15 minutes, even if the maximum number of sessions defined in `history_sessions` has not been reached.

max_ftacl_cache_size

Content Server caches ACL information on objects to evaluate security on the results returned by full-text queries. The `max_ftacl_cache_size` key defines the number of elements cached.

The default value is -1 (no limit set). If the value is set to 0, no security information is cached. The value may be set to any integer greater than -1. It is recommended that you do not change the default value.

max_sessions_heap_size

The `max_sessions_heap_size` key allows you to control the size of a session's memory usage. This number is expressed in bytes.

The default value is -1, meaning that heap will grow as necessary to whatever size the server machine resources will allow, up to a server's addressing limits of 2 GB of memory.

saveasnew_retain_source_group

The `saveasnew_retain_source_group` key controls which default group is assigned to a new object created by a `Saveasnew` method. If the key is set to `T` (TRUE), the new object is assigned the default group of the original (source) object. If the key is set to `F` (FALSE), the new object is assigned the default group of the user who issues the `Saveasnew` method. The default is `F`.

umask (UNIX only)

The `umask` `server.ini` key modifies the default operating system permissions assigned to public directories and files created by Content Server in the server or repository installation. For example, this affects the storage area directories and the full text index directories, as well as the files written to those directories.

Note: If `umask` is not set, the default operating system permissions are `777` for directories and `666` for files.

The `umask` key works similarly to the UNIX `umask` functionality. The value you assign to the key is subtracted from the default permissions to determine the actual permissions assigned to a file or directory. For example, if you set `umask=2`, then the default permissions assigned to directories becomes `775` and the default permissions for files becomes `662`. Or, if you set `umask=20`, then the permissions become `757` for directories and `626` for files.

upd_last_chg_time_from_db

The `upd_last_chg_time_from_db` key is used to ensure that all Content Servers in a clustered environment have timely access to all changes in group membership. The default value is `F` (FALSE). Set `upd_last_chg_time_from_db` to `T` (TRUE) only in an environment where multiple servers run against a repository. In such an environment, set the key to `T` for all running Content Servers.

use_group_address

The `use_group_address` key controls who receives the email notifications when an event is queued to a group. Valid values and their associated behaviors are:

- `0`, which directs the server to send email to each member of the group.

- 1, which directs the server to send email to group's address (identified in the group's `group_address` attribute). If the group has no email address, the server sends notifications to each member of the group.
- 2, which directs the server to send email to each member of the group and to the group address, if it exists.

The default setting is 0.

validate_database_user

The `validate_database_user` key controls whether Content Server validates that the user identified in the `database_owner` key in the `server.ini` file has a valid operating system user account. The default value is T (TRUE), meaning that the existence of a OS account for the database owner is validated.

The default keys

During the server installation procedure, default values are assigned to the keys described in this section. You are not prompted for their values. You can change these values at any time.

client_session_timeout

The `client_session_timeout` key defines how long the server waits for a communication from a client session before disconnecting the session.

The default value is 5 minutes.

concurrent_sessions

The `concurrent_sessions` key controls the number of connections the server can handle concurrently. This number must take into account not only the number of users who will be using Documentum concurrently, but also what kinds of operations those users will be executing. Some operations require a separate connection to complete the operation. For example:

- Issuing a Query method or an Execquery method with the `readquery` flag set to FALSE causes an internal connection request.
- Executing a method using Apply or EXECUTE starts another connection.

- When the agent exec executes a job, it generally requires two additional connections.
- Issuing a full-text query requires an additional connection.

The default value is 100. Consider the number of active concurrent users you expect, the operations you think they will be performing, and the number of methods or jobs you execute regularly, and then modify this value accordingly.

The maximum number of concurrent sessions is dependent on the operating system of the server host machine. For Solaris and AIX platforms, the limit is 1022. For HP-UX, it is 2046. For Windows systems, it is 1024.

database_refresh_interval

The `database_refresh_interval` key defines how often the main server thread (parent server) reads the repository to refresh its global caches. You can raise this value but it cannot be lowered.

The default value is 1 minute.

distinct_query_results

The `distinct_query_results` key tells the server whether to return duplicate rows in queries. `FALSE` tells the server to include duplicate rows in query results. `TRUE` tells the server to return only distinct rows (no duplicates).

If `NOFTDQL` hint is specified for a query and the `distinct_query_results` flag is set to `TRUE`, only unique results are returned by the query. If the identical query is executed in `FTDQL` and the `distinct_query_results` key is set to `TRUE`, the setting of the key is ignored and *all* results of the query are returned, including duplicate results.

The default value is `FALSE`.

enforce_four_digit_year

Set to `TRUE`, the `enforce_four_digit_year` key directs the server to set or display dates using a four digits for the year if the user does not specify a date format.

The default value is `TRUE`.

ignore_client_domain (Windows only)

The `ignore_client_domain` key indicates whether the server ignores the domain passed to it by the client during a connection request. `TRUE` directs the server to ignore the client domain and use the domain specified in `user_auth_target` for all user authentications. `FALSE` directs the server to use the client domain passed in a connection request.

The default value is `FALSE`.

mail_notification

The `mail_notification` key is a Boolean key that controls whether email messages are sent when a work item or an event is queued. If this is set to `FALSE`, the email messages are not sent to the users.

The default is `TRUE`.

method_server_enabled

The `method_server_enabled` key is a Boolean key that controls whether the method server or the Java method server may be used to execute `dm_method` objects. The default value is `T (TRUE)`, meaning that `dm_method` objects that are configured to execute through the method server do so with no further configuration needed. If the `method_server_enabled` is set to `F (FALSE)`, the methods are executed through Content Server.

Note: Enabling the method server alone does not cause `dm_method` objects to be executed by the method server or Java method server. The method objects must also be configured correctly. For more information, refer to [Creating a method object, page 141](#).

method_server_threads

The `method_server_threads` key defines the maximum number of method server worker processes are available to execute method objects. The default (and minimum) value is 5. The maximum value for this key is the value set in the `concurrent_sessions` attribute of the server config object.

preserve_existing_types

When a server starts, it queries the RDBMS to determine if the object type tables are present for all the object types defined in the server.

The default value is TRUE, meaning the server does not dynamically destroy and recreate object type tables for types that are erroneously reported missing by the RDBMS.

If this flag is set to FALSE, the server does dynamically destroy and recreate object type tables for types that are erroneously reported missing by the RDBMS.

rdbms_connect_retry_timeout

The `rdbms_connect_retry_timeout` key determines how long the server tries to connect to the RDBMS. The server attempts to connect every 30 seconds until it is successful or the time-out limit is reached.

The default time-out value is 5 minutes.

server_config_name

The `server_config_name` key identifies which server config object is used to start the server. When you install Content Server, the procedure assigns the name of the repository to the server config object generated for the server.

The default value is the name of the repository.

server_startup_sleep_time

The `server_startup_sleep_time` key specifies the amount of time, in seconds, that Content Server waits before trying to connect to the RDBMS. The time delay allows the underlying RDBMS to start before Content Server attempts to connect.

The default value is 0.

start_index_agents

The `start_index_agents` key indicates whether Content Server starts configured index agent instances at Content Server startup.

The default value is TRUE.

ticket_multiplier

The ticket_multiplier key determines the number of login tickets with server scope allocated in shared memory. The number of tickets allocated by the server is computed as follows:

$$\#tickets = concurrent_sessions * ticket_multiplier$$

The default value is 10.

update_access_date

The update_access_date key indicates whether the r_access_date attribute is updated in the deferred update process. TRUE directs the server to update the attribute as part of the deferred update process. FALSE directs the server not to update the attribute.

The default value is TRUE.

user_auth_case

The user_auth_case key indicates the case (upper, lower, or unspecified) to which the server should convert the client's user name before authenticating the user. Valid settings are:

- upper
- lower
- NULL

The default is NULL, meaning the name is authenticated using the case in which it was entered by the user.

user_auth_target (Windows only)

The user_auth_target key identifies the domain or default LDAP server that Content Server uses to authenticate the client's user name and password.

The default is the domain in which the server resides.

use_estimate_search

The `use_estimate_search` key controls whether users can execute the `ESTIMATE_SEARCH` administration method. The administration method is used to fine-tune `SEARCH` conditions for queries. `FALSE` means that the method will not execute if a user tries to use the method. `TRUE` allows the method to execute.

The default is `F (FALSE)`.

wait_for_connect_timeout

The `wait_for_connect_timeout` key defines how long the server waits for a connection request before starting to process other work, such as deferred updates.

The default value is 10 seconds.

Moving the server executable (UNIX only)

The Content Server installation procedure places the server executable in the `$DM_HOME/bin` directory. If you move the executable to another directory, you must modify the `DM_HOME_CURRENT` variable in the `dm_start_docbase` script or modify the command line in the script that references the executable.

The command line references the executable using a relative path. The script sets the current directory to the directory specified in `DM_HOME_CURRENT` before executing the command line.

If you change `DM_HOME_CURRENT` to the new location of the executable, the script uses that location as the current directory when it executes the command line.

Alternatively, you can replace the `./` in the command line with the full path to the executable's new location. In this case, the script ignores the setting of `DM_HOME_CURRENT`.

Changing default operating system permits on directories and files (UNIX only)

When Content Server creates directories and files in the server installation, it assigns default operating system permissions to those directories and files. The default permissions assigned to directories are `777` and the default permissions assigned to files

are 666. You can change the defaults assigned to public directories and files by setting the `umask` key in the `server.ini` file. Setting `umask` affects all public directories and files created after the key is set.

The `umask` value works similarly to the UNIX `umask` functionality. The value is subtracted from the default permissions to determine the actual permissions assigned to a file or directory. For example, if you set `umask=2`, then the default permissions assigned to directories becomes 775 and the default permissions for files becomes 662. Or, if you set `umask=20`, then the permissions become 757 for directories and 626 for files.

Changing a server's configuration

Depending on what you are changing in the server's configuration, modify either the `server.ini` file or the server config object.

Modifying the `server.ini` file

To change the `server.ini` file, you must have appropriate access privileges for the `%DOCUMENTUM%\dba\config\repository` (`$DOCUMENTUM/dba/config/repository`) directory in which the file resides. Typically, only the installation owner can access this directory.

To modify the `server.ini` file:

1. Open the `server.ini` file.
On UNIX, use the text editor of your choice.
On Windows:
 - a. Choose `Start>Programs>Documentum>Content Server Manager`.
 - b. Select the repositories tab.
 - c. Select the repository associated with the `server.ini` file.
 - d. Click `Edit Server.ini`.
2. Make the changes you want.
3. Save the file.
4. Stop and restart the server to make the changes visible.

Modifying the server config object

The server start up procedure always uses the CURRENT version of the server config object.

To change the server config object, you must have Sysadmin or Superuser privileges.

Use Documentum Administrator or the API to modify the server config object. For instructions on using Documentum Administrator, refer to the Documentum Administrator online help.

To modify the server config object using the API:

1. Begin an API session.
2. Use the Fetch or Checkout method to obtain the server config object from the repository.

```
fetch,c,server_config_object_id
```

```
checkout,c,server_config_object_id
```

If you do not know the object ID of the server config object, you can use the Retrieve method to obtain it:

```
retrieve,c,dm_server_config  
where object_name = 'server_config_name'
```

server_config_name is typically the name of your repository.

3. Use the Set method make the changes you want.
You can use the keyword `serverconfig` to represent the server config object in the Set syntax. For example,

```
API>set,s0,serverconfig,keep_entry_interval  
SET>12  
...  
OK
```

4. Use the Save or Checkin method to save your changes.

```
save,s0,server_config_object_id
```

```
checkin,s0,server_config_object_id
```

You cannot use the `serverconfig` keyword in these methods.

5. Reinitialize the server to make the changes visible.
 - To reinitialize the main server thread and the thread (parent server and the spawned server) for your current session, use the Reinit method. Other current sessions are not affected.

```
reinit,session
```

- To reinitialize just the session server, use the Restart method:

```
restart,session
```

Setting the secure connection mode

When you install a Content Server, two service names and associated ports are defined. One port is a native, or unsecure port, the other is a secure port. (For more information about the service names and their ports, refer to *Content Server Installation Guide*.) During the installation process, you are asked what connection mode you want for the server. There are three valid modes:

- native
- secure
- native and secure

native means that the server listens on an unsecure port. Connection requests from clients that ask for a secure connection will fail.

secure means that server listens only on a secure port. Connection requests from clients that ask for a native connection will fail.

native and secure means that server listens on both a native and a secure port. Connection requests from clients asking for either type of connection are accepted.

To change the secure connection mode for the server, reset the `secure_connect_mode` attribute in the server's server config object. You can use Documentum Administrator, DQL, or the API to reset the `secure_connect_mode` attribute. You must restart the server after resetting the `secure_connect_mode` attribute.

Note: Do not set the mode to secure if you have pre-5.2 clients connecting to the repository. The connection requests from such clients will fail.

Restarting a server

How to restart a server depends on the repository configuration.

If a repository has other active servers, use Documentum Administrator to restart the server. For instructions, refer to the Documentum Administrator online help.

If a repository has only one server or if all servers for a repository are shut down, use the following procedure to restart a server for the repository.

Use the following procedure for servers on a Windows host.

To restart a server using Content Server Manager:

1. Log into the machine where Content Server is installed as the Documentum installation owner.
2. Choose Start>Programs>Documentum>Documentum Content Server Manager.

3. Select the repositories tab.
4. Click Start.

Use the following procedure for servers on a UNIX host.

To restart a server using the startup script (UNIX):

1. Log into the machine where Content Server is installed as the Documentum installation owner.
2. Change to the \$DOCUMENTUM/dba directory.
3. Run the `dm_start_repository` script that references the server you want to start.

Starting additional servers

Note: Do not use the instructions in this section to set up servers for a single-repository distributed configuration. Setting up a single-repository distributed configuration requires more than changing the configuration of the server. Refer to [Chapter 3, Implementing Single-Repository Models](#), in the *Distributed Configuration Guide* for those instructions.

You can start additional servers for a repository on different machines or the same machine. The service name for each server must be unique within the network connecting the machines and that service name must be referenced in the service attribute of the `server.ini` file invoked for the server.

Servers servicing one repository must be all non-trusted servers or all trusted servers. You cannot have trusted and non-trusted servers servicing one repository.

On a Windows host, each server must have its own server config object and `server.ini` file. Creating a new server using Documentum Administrator automatically creates them for you.

On a UNIX host, each server must have its own server config object and `server.ini` file, and start up script. You may also want a shutdown script for each server. Creating the new server using Documentum Administrator automatically creates the server config object, the `server.ini` file, and the start up script. You must manually create the shutdown script.

A server-specific server config object is needed because each server must reference local copies of the `assume user`, `check password`, `change password`, and `secure writer` programs. These programs are located by a server through location objects defined in its server config object. If the servers are in different geographical locations, there are also several other configuration settings you will probably change.

A server-specific `server.ini` file is needed because the file references the server config object. When a server is started, the `server.ini` file is read and the appropriate server config object must be invoked.

Configuration requirements

Note: For information about starting additional servers on existing Content Server hosts, refer to the Appendix titled “Configuring Multiple Servers in the Same Installation” in the *Content Server Installation Guide*.

The following configuration requirements must be met to run multiple servers for the same repository:

- Windows-specific requirements:
 - All machines running a Content Server for a particular repository must be in the same domain.
 - The primary host’s installation account must be in the domain’s administrators group.
 - The program SC.EXE must be installed in %SystemRoot%\System32\SC.EXE. SC.EXE is available on the Windows SDK CD and the Windows NT Resources Kit CD. It is used to create and start services for Documentum servers.
- UNIX-specific requirements:
 - The primary server’s installation account must have sufficient privilege to execute remsh commands.
 - The primary host must be able to resolve target machine names through the /etc/hosts file or DNS.
 - Each target machine must be able to resolve the primary host name through the /etc/hosts file or DNS.
 - If you intend to use Documentum Administrator’s Server Management feature to create and start a new, remote server, there are some additional configuration requirements. Refer to Documentum Administrator online help for details.
- General requirements for all platforms:
 - The Documentum installation accounts for each site must be known to all hosts. It is recommended that each site use the same account as the installation account.
 - The servers must all be non-trusted servers or all trusted servers.

Creating a shut-down script (UNIX only)

Use the following instructions to create a server-specific shut-down script.

To create a `dm_shutdown_repository` script for a new server:

1. Make a copy of an existing `dm_shutdown_repository` script and give the copy a new name.

`dm_shutdown_repository` scripts are found in `$DOCUMENTUM/dba`.

For example:

```
% cd $DOCUMENTUM/dba
% cp dm_shutdown_engrrepository dm_shutdown_devrepository1
```

2. In the new copy, edit the line immediately preceding `shutdown,c,T` by replacing `repository name` with `repository_name.server_config_name`.

The line you want to edit looks like this:

```
./iapi <repository name> -U$DM_DMADMIN_USER -P -e << EOF
```

Use the name of the server config object that you created for the server. For example:

```
./iapi devrepository2.server2 -U$DM_DMADMIN_USER -P -e << EOF
```

3. Save the file.

Communicating with connection brokers

The connection broker is the intermediary between clients and servers for all connection requests. Servers must regularly broadcast, or *project*, connection information to at least one connection broker to be considered active.

Server broadcasts are called checkpoints, and the interval between them is configurable (refer to [Setting the checkpoint interval, page 114](#), for instructions). The connection brokers receiving the checkpoints are called connection broker projection targets.

In addition to the connection information, the server sends each projection target a proximity value. This value describes the server's proximity to the connection broker. The connection broker passes the proximity value to clients so they can decide which server to use.

The connection broker keeps a server's connection information for a length of time called the keep entry interval. This is a configurable interval (refer to [Setting the keep entry interval, page 116](#), for instructions).

Defining connection broker projection targets

Connection broker projection targets are defined in either the server config object or the `server.ini` file. If you define projection targets in the server config object, you can modify the targets, add a new target, or remove a target and implement the changes by simply

reinitializing the server. If you define the projection targets in the server.ini file, you must stop and restart the server to implement the changes.

Note: By default, when you install remote Content Servers, they are configured with default projection targets and proximity values. A Content Server at a remote site is configured to project to its local connection broker and the connection broker at the primary site. Its proximity value for the local connection broker is 9001 and the value projected to the primary site is 9010.

If you define the same connection broker projection target in both the server config object and the server.ini file, the definition in the server config object overrides the definition in the server.ini file.

The Content Server installation procedure defines one connection broker projection target in the server.ini file. The target is the connection broker you specify during the installation procedure. When the installation procedure is complete, you can move the target definition to the server config object if you like.

Definitions in the server config object

The following repeating attributes in the server config object store connection broker projection target definitions:

- projection_targets

The projection_targets attribute contains the name of the host machine on which the connection broker resides.

- projection_ports

The projection_ports attribute contains the port number on which the connection broker is listening.

- projection_proxval

The projection_proxval attribute contains the proximity value that the server projects to the connection broker.

- projection_enable

The projection_enable attribute determines whether the server projects to the connection broker. If it is set to TRUE, the server projects to the connection broker. If it is set to FALSE, the server does not project the connection broker.

- projection_notes

The projection_notes attribute is a place for you to record short notes about the target. The attribute is 80 characters long.

The values at the same index positions across the attributes represent the definition of one connection broker projection target. For example, suppose you want to define a

second projection target for `server_1`. The target connection broker resides on `bigdog`, and you want to project a proximity value of 10 to the connection broker for the server. Set the attributes in `server_1`'s server config object to the following values:

```
projection_targets[1]=bigdog
projection_ports[1]=1489
projection_proxval[1]=10
projection_enable[1]=T
projection_notes[1]=your comments
```

Use Documentum Administrator to modify the server config object to set the attributes. For instructions, refer to the Documentum Administrator online help.

Definitions in the server.ini file

Connection broker projection targets are identified in the `server.ini` file in one or more `[DOCBROKER_PROJECTION_TARGET]` sections.

The format for the first target definition is:

```
[DOCBROKER_PROJECTION_TARGET]
host=connection broker host name
port=connection broker port number
proximity=proximity_value
```

The sections for additional definitions differ only in the addition of an index value to the title of the section:

```
[DOCBROKER_PROJECTION_TARGET_n]
...
```

where n is an integer from 0 to 49. You can include a maximum of 50 sections.

Setting the checkpoint interval

A checkpoint interval defines how often a server broadcasts service information to connection brokers. Checkpoint intervals are defined in the `checkpoint_interval` attribute in the server's server config object. You can set the checkpoint interval using Documentum Administrator.

You can use Documentum Administrator or the API to modify the server config object to set the checkpoint interval. For instructions on using Documentum Administrator to modify the server config object, refer to the Documentum Administrator online help.

Using the API

You can access the API through Documentum Administrator or using IAPI. These instructions describe how to use IAPI. For instructions on accessing the API through Documentum Administrator, refer to the Documentum Administrator online help.

To change the checkpoint interval using IAPI:

1. Start IAPI, connecting to the repository as a user with Sysadmin or Superuser privileges.

2. Fetch the server config object.

```
fetch,c,server_config_object_id
```

If necessary, use the Retrieve method to obtain the server config object ID:

```
retrieve,c,dm_server_config where object_name='server_config_name'
```

3. Use the Set method to change the `checkpoint_interval` attribute.

For example,

```
API>set,c,serverconfig|server_config_id,checkpoint_interval
SET>new_checkpoint_value
...
OK
```

Include the `serverconfig` keyword if you are changing the interval for the server to which the session is connected.

To set the interval for a different server on the same repository, include the object ID of that server's server config object.

new_checkpoint_value is the checkpoint interval, in seconds, that you want to assign to the server.

4. Save the server config object:

```
save,c,server_config_id
```

You cannot use the `serverconfig` keyword in the Save method.

5. Reinitialize the server to make the change effective:

- If you changed the server for the current session, use the following syntax:

```
reinit,c
```

- If you changed another server associated with the repository, use the following syntax:

```
reinit,c,server_config_name
```

server_config_name is the name of the server whose interval you modified.

Setting the keep entry interval

Each connection broker is told how long it can keep a server entry if it does not receive checkpoint broadcasts from the server. This time limit is defined in the `keep_entry_interval` attribute of a server's server config object and is included in the server's checkpoint information. By default, the keep entry interval is set to 24 hours (expressed in minutes).

You can use Documentum Administrator or the API to modify the server config object to set the keep entry interval. For instructions on using Documentum Administrator to modify the server config object, refer to the Documentum Administrator online help.

Using the API

You can access the API through Documentum Administrator or using IAPI. These instructions describe how to use IAPI. For instructions on accessing the API through Documentum Administrator, refer to the Documentum Administrator online help.

To change the keep entry interval using IAPI:

1. Start IAPI, connecting to the repository as a user with Sysadmin or Superuser privileges.
2. Fetch the server config object.

```
fetch,c,server_config_object_id
```

If necessary, use the Retrieve method to obtain the server config object ID:

```
retrieve,c,dm_server_config where object_name='server_config_name'
```

3. Use the Set method to change the `keep_entry_interval` attribute.

For example,

```
API>set,c,serverconfig|server_config_id,keep_entry_interval
SET>new_interval_value
...
OK
```

Include the `serverconfig` keyword if you are changing the interval for the server to which the session is connected.

To set the interval for a different server on the same repository, include the object ID of that server's server config object.

new_interval_value is the keep entry interval, in minutes, that you want to project to the connection broker.

4. Save the server config object:

```
save,c,server_config_id
```

You cannot use the `serverconfig` keyword in the `Save` method.

5. Reinitialize the server to make the change effective:
 - If you changed the server for the current session, use the following syntax:

```
reinit,c
```
 - If you changed another server associated with the repository, use the following syntax:

```
reinit,c,server_config_name
```

server_config_name is the name of the server whose interval you modified.

Defining server proximity

Servers send a proximity value to each connection broker projection target. The proximity value represents the server's physical proximity to the connection broker.

When clients receive server information from a connection broker, by default they choose to connect to the server with the smallest proximity value (representing the closest available server). For example, assume a client gets information about servers A, B, and C. The proximity value is 2 for A, 4 for B, and 5 for C. The client will attempt to connect to server A because that server has the lowest proximity value. If two or more servers have the same lowest value (for example, if both servers A and B have a value of 2) then the client makes a random choice between the servers.

Note: Clients and users can override this default behavior by specifying either a server or host machine (or both) on the `Connect` command line. Refer to [Connect, page 155](#), in the *Content Server API Reference Manual* for details.

We recommend that the proximity values reflect the topology of your installation. For example, if you have three servers and one connection broker, the server closest to the connection broker should project the lowest proximity value to the connection broker. The server farthest from the connection broker should project the highest proximity value to the connection broker.

An individual server that has multiple connection broker projection targets can project a different proximity value to each target.

Guidelines for setting proximity values are:

- Use proximity values in the range of 1 to 999 unless you are setting up content servers for a distributed configuration.
- Any server with a proximity value of 9000 to 9999 is considered a content server and typically will only be used to handle content requests.

For information about content and data servers and how to set them up, refer to [Content-file servers, page 28](#), in the *Distributed Configuration Guide*.

- If you specify a value between 1001 and 8999, the fourth digit (counting from the right) is ignored and only the first three digits are used.

For example, if you define a proximity value of 8245, clients ignore the 8 and only consider 245 the proximity value.

- On Windows platforms, proximity values of 10,000 and over are considered to represent servers in another domain.

Users who want to connect to such servers must specify them by name in the Connect command line.

Shutting down a server

On Windows platforms, you can use Documentum Content Server Manager, the Windows user interface, or the Shutdown method to shutdown a server if it is the only active server for a repository. On UNIX, use the `dm_shutdown_repository` script or the Shutdown method.



Caution: On Windows platforms, using Shutdown method, while possible, is not recommended. The method does not use the Windows service manager to shut down the server and, consequently, may not shut down all relevant processes. Using the Documentum Content Server Manager or the Windows service manager to shut down a server on a Windows host is the recommended procedure.

On either platform, if the repository has multiple active servers, use Documentum Administrator to shut down one of the servers. For instructions, refer to the Documentum Administrator online help.

You must have Sysadmin or Superuser user privileges to stop a server.

The procedures in this section shut down the main server thread (parent server). To stop a session server, refer to [Stopping a session server](#), page 120, for instructions.

Using the `dm_shutdown_repository` script (UNIX only)

The `dm_shutdown_repository` script logs into the specified repository and issues a shutdown request. The script waits 90 seconds before exiting. (If the repository shuts down more quickly, the script exits as soon as the server is down.) This script is useful when you want to shut down the server as part of a program or application, without human intervention.

You must run the script on the machine where the server resides. To invoke the script, issue the command:

```
dm_shutdown_docbase [-k]
```

where *docbase* is the name of the server's repository. The *-k* flag instructs the script to issue an operating-system kill command to stop the server if the shut down request has not been completed in 90 seconds.

Using the Documentum Content Server Manager (Windows only)

To shut down a server using the Content Server Manager:

1. Choose Start> Programs> Documentum> Documentum Content Server Manager.
2. Select the repositories tab.
3. Select the repository from the list of repositories.
4. Click Stop.

Using the Windows user interface (Windows only)

Content Server is installed as a Windows service. You can use the Windows Service Control Panel to stop the server.

To stop a server using the Service Control Panel:

1. Double-click the Control Panel icon in the Main group.
The Control Panel window appears.
2. Double-click the Services icon in the Control Panel window.
The Services window appears.
3. Select the server.
4. Click Stop.

Using the Shutdown method

Using the Shutdown method provides you with additional options. The Shutdown method can:

- Shut the server down immediately, without waiting for currently connected users to finish

- Shut the server down after all sessions finish their current transaction
- Direct the connection broker to delete its information about the server

These options are implemented by two optional arguments to the Shutdown method:

```
shutdown, session[, immediate][, delete_entry]
```

By default, the method waits until all current transactions are finished before stopping the server. If you want to stop the server immediately, issue the method with the *immediate* argument set to T (TRUE). For example:

```
shutdown, session, T
```

When you shut down a server using the Shutdown method, the server sends the connection broker a message that it is shutting down. If you intend to restart the server, you want the connection broker to retain the information it has about the server. However, if you are shutting down the server permanently, you want the connection broker to remove information about the server. To accomplish this, set the *delete_entry* flag to T (TRUE). For example:

```
shutdown, session, , T
```

Note: Even though the *immediate* argument is not set, the comma is included as a place-holder. Arguments to API methods are positionally referenced. Even if you do not include the argument, you must include its comma.

Both of these optional arguments are FALSE by default.

To shut down a server using the Shutdown method:

1. Log in to the repository associated with the server you want to shut down.
2. Issue the Shutdown method.

For example, the following method shuts down the server gracefully and does not delete its entry in the connection broker:

```
API>shutdown, s0
```

This examples shuts down the server immediately and deletes the server's entry in the connection broker:

```
API>shutdown, s0, T, T
```

Stopping a session server

Use the Kill method to shut down a session server. To use Kill, you must have Sysadmin or Superuser privileges. You also need the session ID of the session you want to stop. You can obtain this ID using the LIST_SESSIONS or SHOW_SESSIONS administration method.

You can terminate a session in one of three ways:

- The default kill
The default kill provides the least disruption to the end user. The targeted session terminates when it has no more open transactions and no more open collections. The client remains functional.
- An After current request kill
An after current request kill provides a safe and more immediate means of terminating a session. If the session is not currently executing a request, the session is terminated. Transactions may be rolled back and collections may be lost.
- An Unsafe kill
An unsafe kill provides a means for terminating a session when all other techniques have failed. Use this option with caution. It can result in a general server failure.

The method also lets you send a message to the session user. Refer to [Kill, page 295](#), in the *Content Server API Reference Manual* for details.

You can execute a Kill method using the API facility in Documentum Administrator or through IAPI. For instructions on using Documentum Administrator, refer to the Documentum Administrator online help.

To shut down a session server using IAPI:

1. Start IAPI, connecting to the repository as a user with Sysadmin or Superuser privileges.
2. Issue the Kill method.
 - To terminate the session after all transactions are complete and all collections are closed, use:
`kill,c,target_session_id`
 - To terminate the session after the current transaction is completed, use:
`kill,c,target_session_id,after current request`
 - To terminate the session immediately, without waiting for transactions to be completes or collections to be closed, use:
`kill,c,target_session_id,unsafe`
3. If you used an unsafe kill, monitor the server.

To monitor the server, try to connect, look at some documents, and check the server log. If the server appears to be compromised, shut it down and restart it.

Server log files

Each Content Server maintains a log file. By default, the log file is stored in %DOCUMENTUM%\dba\log (\$DOCUMENTUM /dba/log) and named

serverconfig_name.log, where *serverconfig_name* is the name of the server config object used by the Content Server. The default location is defined in a location object named "log" that is referenced by the *log_location* attribute in the server config object. The location object is created and the attribute set by *headstart.ebs*.

You can override both the name and the location of the file by setting the *-logfile* parameter on the server start-up command line.

The server appends to the log file so long as the server is running. If the server is stopped and restarted, the file is saved and another log file started. The saved log files are named using the following format:

On Windows platforms: *serverconfig_name.log.save.mm-dd-yy_hh.mi.ss*

On UNIX platforms: *serverconfig_name.log.save.mm.dd.yyyy.hh.mi.ss*

Server load balancing and failover

When your installation has a large number of users or there is a lot of activity in the repository, you may want to start multiple servers to spread the load. Starting multiple servers will also allow graceful failover if a particular server stops for any reason.

The servers used for load balancing must project identical proximity values to any given connection broker. In that way, when a client DMCL determines which server, it will randomly pick one of the servers. If the values are different, the DMCL will always choose the server with the lowest proximity value.

If a Content Server stops and there are additional servers running against the repository with proximity values less than 9000, the client library, with a few exceptions, will gracefully reconnect any sessions that were connected to the stopped server to one of those servers. The exceptions are:

- If the client application is processing a collection when the disconnection occurs, the collection is closed and must be regenerated again when the connection is re-established.
- If a content transfer is occurring between the client and server, the content transfer must be restarted from the beginning.
- If the client had an open explicit transaction when the disconnection occurred, the transaction was rolled back and must be restarted from the beginning.
- If the original connection was started with a single-use login ticket or a login ticket scoped to the original server, the session cannot be reconnected to a failover server because the login ticket may not be re-used.

If the additional servers known to a session's connection broker do not have the same proximity value, the client library will choose the next closest server for failover. Sessions cannot failover to a Content Server whose proximity is 9000 or greater. Content Servers

with proximities set 9000 or higher are called content-file servers. Remote Content Servers installed at remote, distributed sites are configured as content-file servers by default.

Note: A client session can only fail over to servers that are known to the connection broker used by that session. To ensure proper failover, make sure that Content Servers project to the appropriate connection brokers and with appropriate proximity values.

Clearing the server common area

To remove all files from the server common area:

1. Stop Content Server.
2. On Windows, edit the Content Server service to add `-oclean` to the end of the command line.
3. On UNIX, add the `-oclean` argument in the `dm_start_docbase` command line.
4. Restart the server.

Adding additional servlets to the Java method server

To use the HTTP_POST administration method, you must write and install the servlet or servlets that handles those calls.

To implement a new servlet:

1. Write the servlet.
2. Install the servlet on the Java method server.
3. Update the server config object for each server from which HTTP_POST methods might originate.

Add the new servlet's name to the `app_server_name` attribute and the servlet's URI to the `app_server_uri` attribute.

Use Documentum Administrator to modify the server config object. For instructions on using Documentum Administrator, refer the the Documentum Administrator online help.

4. Check Re-initialize.
5. Click Check In.

Configuring the workflow agent

The workflow agent controls the execution of automatic activities in a workflow. The agent is comprised of a master repository session and one or more worker sessions. The master session is quiescent until the workflow agent is notified by Content Server that an activity has been created or until the sleep interval expires. At that time, the master session queries the repository for information about the activity or activities and assigns the waiting activity to a free worker session.

You can change the workflow agent's defaults by:

- Changing the number of worker sessions
- Changing the sleep interval
- Disabling the workflow agent

In addition, you can trace workflow agent operations. [Tracing the workflow agent, page 125](#), describes how tracing is started.

Changing the number of worker sessions

The number of worker sessions available to execute automatic activities is controlled by the `wf_agent_worker_threads` attribute value in the server config object. By default, this value is set to 3. You can reset the value to any positive number to a maximum of 1000.

Use Documentum Administrator to change the value. You must stop and restart Content Server for your change to take effect. Reinitializing the server does not make the change effective.

Changing the sleep interval

The sleep interval determines how long the master session sleeps after querying the repository for activity information in the absence of a notification from Content Server. If the sleep interval expires without a notification, the master session wakes up and queries the repository even though it has not received a notification from Content Server. The default sleep interval is 5 seconds.

The sleep interval is controlled by the `wf_sleep_interval` attribute in the server config object. The value is interpreted in seconds. Use IAPI or IDQL to change the value. You must stop and restart Content Server for your change to take effect. Reinitializing the server does not make the change effective.

Disabling the workflow agent

Disabling the workflow agent stops the execution of automatic activities. To disable the workflow agent, set the `wf_agent_worker_threads` attribute in the server config object to 0. Use Documentum Administrator to set the attribute. You must stop and restart Content Server for your change to take effect. Reinitializing the server does not make the change effective.

Tracing the workflow agent

By default, tracing for the workflow agent is turned off. You can turn it on by either:

- Adding the `-otrace_workflow_agent` argument on the server startup command line
- Executing a `SET_OPTIONS` administration method with the `-trace_workflow_agent` option specified

If you add the argument to the server startup command line, you must restart the server to start the tracing. However, workflow agent tracing is turned on automatically when the server starts.

If you use `SET_OPTIONS`, tracing starts immediately. It is not necessary to restart the server. However, if you stop and restart the server, you must reissue the `SET_OPTIONS` administration method to restart the tracing.

For instructions on adding the argument to the startup command line, refer to [Server start-up tracing options, page 543](#). For instructions on using `SET_OPTIONS`, refer to [SET_OPTIONS, page 312](#), in the *Content Server Administrator's Guide*.

The trace messages are recorded in the server log file.

Recovering automatic activity work items on Content Server failure

If a Content Server fails, the workflow agent associated with that server also stops. In such cases, there may be work items that were claimed by the workflow agent prior to the failure but not yet processed. If you can restart the Content Server, the workflow agent will recognize these work items and process them after the restart. However, if you cannot restart the Content Server, you need to remove the workflow agent's claim to those work items so that another workflow agent can claim and process them.

A workflow agent claims a work item by setting that work item's `a_wq_name` attribute to the name of the server config object of the Content Server with which the agent is associated. To remove a workflow agent's claim to a work item, use the

RECOVER_AUTO_TASKS administration method to reset that attribute. The method resets the a_wq_name attribute of the work item to empty, which allows another workflow agent to claim that work item. The method operates only on the work items claimed by a workflow agent associated with a specified server. Other work items claimed by agents associated with other servers are not affected.

Methods and Jobs

This chapter describes how to create and execute methods and jobs. The chapter includes the following topics:

- [Introducing methods, page 127](#)
- [Execution agents, page 128](#)
- [Choosing the execution agent, page 129](#)
- [Tracing options for methods, page 131](#)
- [Defining the java.ini file \(UNIX only\), page 132](#)
- [Enabling the dmbasic method server, page 133](#)
- [Configuring the worker threads in the dmbasic method server, page 133](#)
- [Implementing a method, page 134](#)
- [Executing a method on demand, page 142](#)
- [Creating jobs and job sequences, page 143](#)
- [Managing jobs, page 151](#)

Introducing methods

Methods are executable scripts or programs that are represented by method objects in the repository. The script or program can be a Docbasic script, a Java method, or a program written in another programming language such as C++. The associated method object has attributes that identify the executable and define command line arguments, and the execution parameters.

Methods are executed by issuing a DO_METHOD administration method or using a job. Using a DO_METHOD allows you to execute the method on demand. Using a job allows you to schedule the method for regular, automatic execution. For information about creating jobs, refer to [Creating jobs and job sequences, page 143](#).

The executable invoked by the method can be stored in the file system or as content of the method object. If the method is to be executed by the Java method server, you must store it in %DOCUMENTUM%\dba\java_methods (\$DOCUMENTUM/dba/java_methods).

You can execute a method using the method server, the Java method server, or Content Server. [Execution agents, page 128](#), describes each of these agents and [Choosing the execution agent, page 129](#), provides some guidelines for choosing which to use.

Execution agents

The execution agents are the server processes that are capable of executing methods. There are three execution agents: the dmbasic method server, the Java method server, and Content Server.

dmbasic method server

The dmbasic method server is a separate process that is installed with Content Server and resides on the same host. It is enabled by default. After it is started, it runs continuously. If you stop Content Server, the method server is also stopped. If the method server stops, Content Server restarts it automatically. For instructions on enabling and disabling the method server, refer to [Enabling the dmbasic method server, page 133](#).

The dmbasic method server uses a method execution queue to manage methods submitted for execution. When you direct a method to the method server, Content Server places a method execution request on the bottom of the method execution queue. The method server reads the queue, executing the request at the top of the queue. The method server has a configurable number of worker threads to execute requests. The maximum number of requests the queue can contain is the number of threads times 50. For example, if the method server is configured to use 5 threads, then the method execution queue can contain 250 requests. For instructions on configuring the number of threads, refer to [Configuring the worker threads in the dmbasic method server, page 133](#).

The method server uses connection pooling. This is true regardless of whether connection pooling is enabled for the Documentum Server.

Java method server

Documentum provides the Apache Tomcat Java method server as a component of the Content Server installation. In addition to Tomcat, Documentum provides a servlet to execute methods called by DO_METHOD. The servlet is referenced in the server config's

app_server_name attribute as do_method. (Refer to *Content Server Installation Guide* for more information about how to configure Tomcat and the servlet.)

The Java method server resides on the same host as Content Server. It is not started or stopped automatically when Content Server starts and stops. It must be manually started and stopped. On Windows, the installation program creates a service called **Documentum Java Method Server** that you can use to start and stop the Java method server. On UNIX, the installation program creates a script called startup.sh in \$DM_HOME/tomcat/bin that you can use to start the server and a script called shutdown.sh that you can use to stop the server.

The programs launched with DO_METHODs sent to the Java method server must be Java methods stored in %DOCUMENTUM%\dba\java_methods (\$DOCUMENTUM/dba/java_methods). When a DO_METHOD method directed to the Java method server is issued, it generates an internal HTTP_POST request. The Java method server invokes the do_method servlet to service the request.

Note: The Java method server can also be used execute Java methods that are not associated with a method object. To do this, you must use an HTTP_POST administration method to send the request to the Java method server. For details, refer to [HTTP_POST, page 228](#), of the *Content Server DQL Reference Manual*.

Content Server

Documentum Content Server is the default execution agent for a method if you do not set the method's use_method_server attribute to TRUE to direct the execution to the method server or Java method server.

Choosing the execution agent

Which execution agent you choose to execute a method depends on the language of the method's program and where the program is stored. You can:

- Use the method server to execute Docbasic scripts.
- Use Content Server to execute Docbasic scripts or programs in any language.
- Use a Java method server to execute Java methods that are installed on the Java method server.

To execute Java or DFC calls using the method server, the Java or DFC calls must be called in a Docbasic program.

You cannot use the Java method server to execute Docbasic scripts or any program other than Java methods installed on the Java method server.

Content Server is the default execution agent for methods. If you do not specifically direct a method to use the method server or the Java method server, Content Server executes the method. Methods are directed to the method server or Java method server by setting the method object's `use_method_server` attribute to `TRUE` and setting the remaining attributes appropriately. (Refer to [Creating a method object](#), page 141, for instructions.)

Performance considerations

Using either the method server or the Java method server, whenever possible, improves the performance of method execution.

To execute a method, Content Server generates a new process to execute the method. That process opens a new session with the repository, which in turn opens another session with the RDBMS.

Using the method server avoids the overhead of the additional repository session and RDBMS session. The method server runs continuously, so no new process is created to execute the method. The first time the method server executes a method, it opens a repository session and an associated RDBMS session. Thereafter, the method server uses connection pooling. Using connection pooling removes the need to open new repository and RDBMS sessions for subsequent method executions.

Using the Java method server provides similar performance benefits.

Security considerations

Security is only a consideration when you are executing a method on the Java method server. When executing on the Java method server, there are two security issues:

- Determining the origin of the generated `HTTP_POST` request generated by the `DO_METHOD`.
- Login without passwords (this is only possible on Windows platforms)

To resolve the first issue, the invoked servlet ensures that the generated request comes from a machine that hosts a Content Server by checking the IP address of the sender against a list of repositories. This list is set up when Tomcat is installed and configured. When the Java method server receives a request, it issues a DFC `getServerMap()` call to obtain a list of all servers servicing the repository. From that list, it determines whether the IP address of the machine that originated the request is a Content Server host. If the sender's IP address doesn't match the IP address of a Content Server host, the request fails.

The second issue occurs because the Tomcat Java method server runs as the Content Server installation owner. Consequently, the servlet it invokes to execute `DO_METHOD`

calls also runs as the installation owner. On Windows platforms, the current operating system user is allowed to log in to the repository without providing a password. Consequently, a servlet or an invoked Java method can log into the repository with superuser privileges without providing a password.

If you write a method that logs in in that manner, you may want to ensure that the actual user who issues the DO_METHOD to invoke the method has the appropriate privileges to execute the program as a superuser. To do this, send the current user's name and a login ticket to the method in the arguments. The method can use these to log in and check the privileges of the user before connecting as the current operating system user.

Tracing options for methods

You can trace the DO_METHOD administration method, the program execution, or both.

To trace the DO_METHOD method, set the trace_launch attribute of the method object or the TRACE_LAUNCH argument on the DO_METHOD command line. If both are set, the setting on the command line overrides the attribute setting. The trace information, up to 2047 characters, includes the command executed to invoke the method and messages indicating the success or failure of the invocation. The trace information generated by TRACE_LAUNCH is stored in the repository log file.

To trace method execution in the method server or Content Server, set the trace_method_server server flag. You can set this flag using the SET_OPTIONS administration method or on the server startup command line. Tracing information generated by Content Server is stored in the server log file. Tracing information generated by the method server is stored in the method server log file. The log file is stored in

```
%DOCUMENTUM%\dba\log\<repository_id>\MethodServer\MethodServer\server_config_name.log
```

or

```
$DOCUMENTUM/dba/log/<repository_id>/MethodServer/MethodServer/server_config_name.log
```

The log file is created when the method server is started. If a method server log file currently exists, it is saved with a time stamp and a new log file is started.

To trace method execution on the Java method server, set the trace parameter in the web.xml file to true. For example:

```
<init-param>
  <param-name>trace</param-name>
  <param-value>t</param-value>
</init-param>
```

The web.xml file is found in %DM_HOME%\tomcat\webapps\DmMethods\WEB-INF\web.xml (\$DM_HOME/tomcat/webapps/DmMethods/WEB-INF/web.xml).

The log files generated by the Java method server are stored in %DM_HOME%\tomcat\logs (\$DM_HOME/tomcat/logs).

Defining the java.ini file (UNIX only)

On UNIX, both Content Server and the method server require a java initialization file to execute Java or DFC calls in Docbasic scripts. The initialization file, called java.ini, identifies the location of the runtime Java and shared libraries. The java.ini file is created and installed by default when you install Content Server. The format of the file is:

```
java_library_path=full_path_libjava.so      #required
java_version=java_runtime_version          #required
java_classpath=classpath_or_invoked_java_class #required
java_alias_file=full_path_to_dfc_aliases   #optional
java_disabled=Boolean                      #False by default
```

java_library_path

The java_library_path setting must be the full path to the java shared library, libjava.so.

java_version

The valid value for java_version is 1.3.1 or higher.

java_classpath

All the invoked Java classes must be included in this classpath setting. If they are not, a runtime error occurs. If the Docbasic methods invoke the DFC class methods, the classpath must include the dfc.jar file. Paths specified in classpath are separated by semicolons (;). For example:

```
java_classpath=<$dm_home>/classes/com/documentum/dfc.jar:/usr/java/bin/jre
```

java_alias_file

This specifies the full path to the `dfc.aliases` file installed with the DFC. This file is read and parsed by the Docbasic engine. If you include this in the `java.ini` file, you can declare variables with DFC types. If not, you must declare any DFC-allocated object as “Object”. To embed DFC in Docbasic, you must include this parameter.

java_disabled

Setting this flag to TRUE disables the Java VM. When the flag is TRUE, Docbasic methods cannot call Java. The flag is FALSE by default.

Enabling the dmbasic method server

The method server is installed and enabled automatically when you install Content Server.

To enable or disable the method server:

1. Open the `server.ini` file in a text editor.
2. To enable the method server, set the `method_server_enabled` key to T.
3. To disable the method server, set the `method_server_enabled` key to F.
4. Save the `server.ini` file.
5. Restart Content Server.

If the host is a UNIX machine, you must also ensure that the `java.ini` file has been properly configured and installed before using the method server. [Locating the java.ini file \(UNIX only\)](#), page 135, contains information about the `java.ini` file.

Configuring the worker threads in the dmbasic method server

The threads are the processes that execute the requests on the dmbasic method server’s execution queue. By default, there are five threads. You can reset this number to a maximum equal to the number of allowed concurrent sessions on the Content Server.

To change the number of threads:

1. Open the server.ini file in a text editor.
2. Reset the value in the method_server_threads key.
3. Save the server.ini file.
4. Restart the server.

Implementing a method

There are two basic steps to implement a method:

1. Create the script or program.
2. Create the method object.

The details of each step depend on the program language chosen for use and the execution agent you want to use to execute the method. There are three possible execution agents: Content Server, the dmbasic method server, and the Java method server.

For guidelines about creating a method for execution by Content Server or the dmbasic method server, refer to [Creating a method to be executed by Content Server or the dmbasic method server, page 134](#).

For guidelines about creating a method for execution by the Java method server, refer to [Creating a method to be executed by the Java method server, page 138](#).

Creating a method to be executed by Content Server or the dmbasic method server

Use the following guidelines when creating a method to be executed by Content Server or the dmbasic method server.

General guideline for the script or program

If the program starts a repository session, it is recommended that the program call an explicit Disconnect when the session is finished.

Script or program return values for workflow methods

The Content Server facility that executes automatic activities examines the return value for the executed script or program. If the return value is 0, the facility assumes that the execution was successful. If the return value is any value other than 0, the facility assumes that there was an error in the execution and moves the associated work item to the paused state and places it on the supervisor's queue.

We strongly recommend that the script or program associated with an automatic activity return a value other than zero if any error occurs during the execution of the script or program.

Calling Java or DFC in a Docbasic script

If you want to call Java or DFC methods in a Docbasic script, use the following guidelines.

User account (UNIX only)

On UNIX, the user account under which a Docbasic script calls Java or the DFC must have \$DM_HOME/bin in the account's shared library path, LD_LIBRARY_PATH.

dmbasic executables (UNIX only)

Content Server ships with three versions of the dmbasic executable: dmbasic, dmbasic_shr, and dmbasic_noshr.

dmbasic is the standard interface and uses the shared library version of the DMCL. dmbasic_shr is the same as dmbasic and is provided for backwards compatibility. dmbasic_noshr uses a statically linked DMCL. The dmbasic_noshr executable should not be used for executing Docbasic scripts that call DFC.

Locating the java.ini file (UNIX only)

The method server and Content Server use a java.ini file to find the location of the Java runtime and the shared libraries. The method server looks for the file in the following places, in the order listed:

- In the \$DM_JAVA_CONFIG environment variable. The value must be a full-path specification for the file, including the file's name.
- The current directory

- \$DM_HOME/bin

The Content Server looks first for a -j argument on the dmbasic command line. If the -j argument is not found, it uses the same algorithm as the method server to find the java.ini file.

To include the -j argument on the dmbasic command line for a method executing on the Content Server, use the following format:

```
-j init_file_location
```

where *init_file_location* is either a full or relative path to the java.ini file.

For example:

```
dmbasic -entrypoint -j init_file_location
```

Sample code

Java methods are accessed in Docbasic through the Docbasic CreateObject method, specifying the object class to instantiate. After the object is created, its properties and methods are accessed using the dot syntax (*object.property=value*).

Here is sample code that invokes Java:

```
Declare Function SetupSession (ByVal db As String, ByVal ddo As String,
ByVal pass As String) As Object
Declare Sub CreateDocument
Declare Sub Disconnect

Dim session As Object

Sub Entry_Point (ByVal docbase As String, ByVal docowner As String,
ByVal password As String)

    On Error Goto GAFF

    Dim clientx As Object
    Dim client As object
    Dim logininfo As Object

Set session = SetupSession(docbase,docowner,password)
    Call CreateDocument
session.disconnect
    Exit Sub

GAFF:
    Print "An error has occurred."
    Print "The error number is " & Err() & "."
    Print "The error message is: " & Error$ & "."
    Exit Sub

End Sub

Function SetupSession (ByVal db As String, ByVal ddo As String,
ByVal pass As String) As Object

On Error Goto GAFF
```

```

Set clientx = CreateObject("java:com.documentum.com.DfClientX")
  Set logininfo = clientx.getLoginInfo
  logininfo.setUser ddo
  logininfo.setPassword pass
  Set client = clientx.getLocalClient
  Set SetupSession = client.newSession(db,logininfo)
  Exit Function

GAFF:
  Print "An error has occurred."
  Print "The error number is " & Err() & "."
  Print "The error message is: " & Error$ & "."
  Exit Function

End Function

Sub CreateDocument

  Dim dmdoc As Object

  On Error Goto GAFF

  Set dmdoc = session.newObject("dm_document")
  doc_id$ = dmdoc.getObjectId().ToString
  Print Mid(doc_id,1,2)
  If Mid(doc_id,1,2) <> "09" Then
    Print "ERROR: The docbase id " & doc_id & " is not correct."
    Stop
  End If

  Print "Created a document."
  dmdoc.setObjectName "dfcdoc1"
  Print "Set the document name."
  content$ = ".\dfcdoc.ebs"
  Dim emptystr As String
  dmdoc.setContentType "text"
  dmdoc.setFile content
  Print "Assigned a content to the document."
  dmdoc.Save
  Print "Saved the document."
  Exit Sub

GAFF:
  Print "An error has occurred."
  Print "The error number is " & Err() & "."
  Print "The error message is: " & Error$ & "."
  Exit Sub

End Sub

```

Recording the output

To record the output of the script or program, set the SAVE_RESULTS argument to TRUE on the DO_METHOD command line.

Setting method object attributes for Content Server execution

To execute the method using Content Server, the `use_method_server` attribute must be set to F (FALSE). The remaining attributes are set as needed. For example, if the program is written in Docbasic, then `method_type` and `method_verb` are set to `dmbasic`. ([Method, page 305](#), in the *EMC Documentum Object Reference Manual* lists the attributes of a method object and their valid settings.)

Note: Setting `method_type` to `dmbasic` directs Content Server to add `-f` to the beginning of the file name when it executes the method and to pass all arguments specified on the `DO_METHOD` command line to the program. If `method_type` is not set correctly, the method will not execute correctly.

For instructions on how to create a method object, refer to [Creating a method object, page 141](#).

Setting method object attributes for dmbasic method server execution

To execute a method using the `dmbasic` method server, set the method attributes as follows:

- Set `method_type` attribute to `dmbasic`.
- Set `use_method_server` attribute to T (TRUE).
- Set `run_as_server` attribute to T (TRUE).

For instructions on how to create a method object, refer to [Creating a method object, page 141](#).

Creating a method to be executed by the Java method server

Use the following guidelines when creating a method to be executed by the Java method server.

Note: EMC Documentum does not provide basic support for resolving problems encountered when creating or executing custom Java methods or classes. For help, contact Developer Support or Documentum Professional Services.

General guideline for the script or program

If the program starts a repository session, it is recommended that the program call an explicit Disconnect when the session is finished.

Script or program return values for workflow methods

The Content Server facility that executes automatic activities examines the return value for the executed script or program. If the return value is 0, the facility assumes that the execution was successful. If the return value is any value other than 0, the facility assumes that there was an error in the execution and moves the associated work item to the paused state and places it on the supervisor's queue.

We strongly recommend that the script or program associated with an automatic activity return a value other than zero if any error occurs during the execution of the script or program.

Recording the output

If you are executing the DO_METHOD on the Java method server, Documentum recommends that you include the following interface in the method's program. Including this interface is required if you want to save the response to the repository in a document. You can also use this interface to capture error or trace messages from the Java method or servlet.

```
package com.documentum.mthdservlet;
import java.io.OutputStream;
import java.util.Map;

/**
 * Interface for Java Methods that are invoked by the
 * Documentum Content Server.
 */

public interface IDmMethod
{
    /**
     * Serves as the entry point to a Java method (installed in Tomcat) executed by
     * the Content Server DO_METHOD apply method.
     *
     * @param parameters A Map containing parameter names as keys and parameter values
     * as map values. The keys in the parameter are of type String.
     * The values in the parameter map are of type String array.
     * (This map corresponds to the string ARGUMENTS passed by the
     * DO_METHOD apply call.)
     */
}
```

```
* @param output      OutputStream to be sent back as the HTTP response content,  
*                    to be saved in the repository if SAVE_RESULTS was set to TRUE  
*                    in the DO_METHOD apply call.  
*                    NOTE: This output stream is NULL if the DO_METHOD was  
*                    launched asynchronously. Always check for a null  
*                    before writing to the OutputStream.  
*/  
  
public void execute(Map parameters, OutputStream output) throws Exception;  
}
```

Storing Java methods

You must store methods to be executed by the Java method server in
%DOCUMENTUM%\dba\java_methods (\$DOCUMENTUM/dba/java_methods).

Setting method object attributes for Java method server execution

Methods that you intend to execute on an Java method server must have the following attribute values:

- The method_type attribute must be set to Java.
- The use_method_server attribute must be set to T (TRUE).
- The method_verb attribute must be set to a fully qualified class name of a Java implementation class.

For example: com.documentum.services.myAutoMethod

- The run_as_server attribute must be set to T (TRUE).

Note: UNIX users who are authenticated against a Windows domain cannot execute methods under their own accounts. All methods executed by such users must be run with run_as_server set to TRUE.

For instructions on how to create a method object, refer to [Creating a method object](#), page 141.

Additional guidelines for the Java method server

If you wish to use the Tomcat application server installed with Content Server as your Java method server, ensure that Tomcat is listening on the host and port number identified in the server's server config object. The app_server_name attribute in the server config object identifies the application servers recognized by Content Server. The value "do_method" in that attribute represents the Tomcat server. The value in the

corresponding index position in the `app_server_uri` attribute contains the host and port number for the Tomcat application server.

Creating a method object

You must have Sysadmin or Superuser privileges to create method objects. Because method objects are a subtype of the SysObject object type, it is not necessary to reinitialize Content Server after you create a new method object.

Methods are typically created using Documentum Administrator. It is also possible to create a method object using DQL. For instructions on using Documentum Administrator, refer to the Documentum Administrator online help. [Using DQL, page 141](#), describes how to use DQL to create a method object.

The values you set in many method attributes depend on how you intend to execute the method. For guidelines, refer to [Creating a method to be executed by Content Server or the dmbasic method server, page 134](#), and [Creating a method to be executed by the Java method server, page 138](#). (For a list of the attributes of a method object, refer to [Method, page 305](#), in the *EMC Documentum Object Reference Manual*.)

If you are creating a method for a job that will be executed in a job sequence, refer to [Defining success return codes and success status, page 142](#), for information about setting the `success_return_codes` and `success_status` attributes.

Using DQL

Use a DQL CREATE...OBJECT statement to create a method object. The syntax is:

```
CREATE dm_method OBJECT
SET attribute_name[[index]]=value
[,SETFILE filepath CONTENT_FORMAT=format_name]
{,SETFILE filepath PAGE_NO=page_number}
```

Refer to [Method, page 305](#), in the *EMC Documentum Object Reference Manual* for information about the attributes.

To add the content file containing the script or procedure using the SETFILE clause in the CREATE...OBJECT statement, you must have Superuser privileges. Additionally, the content file must be stored on the host on which Content Server is running. (Refer to [Create...Object, page 67](#), in the *Content Server DQL Reference Manual* for complete information about using the SETFILE clause.)

Use an API method (Setfile or Setcontent) to add the content file if you do not have Superuser privileges or if the files are stored on a client machine.

Defining success return codes and success status

The methods executed by jobs included in a job sequence must have a defined value for the `success_return_codes` attribute or the `success_status` attribute or both. The `dm_run_dependent_jobs` method, which invokes sequenced jobs, uses one or both of those attributes to determine whether an invoked job completed successfully.

The `dm_run_dependent_jobs` method compares the values in the `success_return_codes` attribute and the `success_status` attribute to the values in the job's `a_last_return_code` and `a_current_status`. The value in `a_last_return_code` is compared to the value or values in `success_return_codes`, and the value in `a_current_status` is compared to the value in `success_status`.

If you set only `success_return_codes`, `dm_run_dependent_jobs` compares the value in `a_last_return_code` to the value or values in `success_return_codes`, and any value in `a_current_status` is ignored. If you set only `success_status`, `dm_run_dependent_jobs` compares the value in `a_current_status` to the value in `success_status` and ignores the value in `a_last_return_code`. If the values in the comparison operation match, `dm_run_dependent_jobs` considers the invoked job to have completed successfully.

If both `success_return_codes` and `success_status` are set, then `dm_run_dependent_jobs` compares each to its corresponding job attribute. If either comparison fails, `dm_run_dependent_jobs` considers the invoked job to have failed. If both comparisons succeed, then `dm_run_dependent_jobs` considers the invoked job to have succeeded.

Note: The agent exec process sets the job's `a_last_return_code` attribute. The attribute is set to the value returned by the job's method. The job's `a_current_status` attribute must be set directly by the invoked method.

Executing a method on demand

Typically, methods are executed on demand through Documentum Administrator. For instructions, refer to the Documentum Administrator online help.

You can also use a DQL EXECUTE statement or an Apply method to issue a `DO_METHOD` call to execute a method. [DO_METHOD](#), page 197, in the *Content Server DQL Reference Manual* describes how to use `DO_METHOD`. [Execute](#), page 96, in the *Content Server DQL Reference Manual* describes how to use the EXECUTE statement. [Apply](#), page 92, in the *Content Server API Reference Manual* describes how to use the Apply method.

Creating jobs and job sequences

This section contains information about jobs and job sequences, how to create them and how to set their schedules. The following topics are included:

- [Introducing jobs, page 143](#)
- [Introducing job sequences, page 143](#)
- [The agent exec process, page 146](#)
- [Creating a job, page 146](#)
- [Creating a job sequence, page 147](#)
- [Scheduling jobs, page 148](#)
- [Passing arguments, page 149](#)
- [The run_now attribute, page 150](#)

Introducing jobs

Jobs automate method execution. Jobs are stored in the repository as dm_job objects. The attributes of a job object reference an associated method object and define an execution schedule. The methods are executed automatically on the schedule specified in the job. You can schedule jobs to be executed individually or in a job sequence.

Jobs are a useful, easy way to automate tasks that you perform regularly. Documentum provides several system administration tools that are implemented as jobs. These tools perform a variety of common system administration tasks, such as removing old renditions and providing warnings when disk space for content runs low. (Refer to [Chapter 12, Tools and Tracing](#), for a list and description of the tools.)

Jobs are created using Documentum Administrator. You must have Sysadmin or Superuser privileges to create a job. Refer to [Creating a job, page 146](#), for instructions.

Introducing job sequences

A job sequence is a set of one or more jobs that are executed in a user-defined order. You can put replication jobs and custom jobs of any user-defined type in a sequence. You cannot put system-defined jobs, other than replication jobs, in a job sequence. The jobs can reside in different repositories, but all must reside in repositories at the 5.3. level or higher.

Sequences are an effective way to handle replication jobs that might overlap in their operations because they allow you to define the order in which the jobs must be run and no job is run until all of its predecessors complete successfully.

For example, perhaps you have two replication jobs with the same source and target repositories. To ensure that the two jobs do not try to load the replicas into the target repository at the same time, you can put both in a job sequence and they are run one after the other, in the order you define.

A job sequence can contain any number of jobs, and the jobs can reside in different repositories. However, you cannot put a job that requires multiple invocations, such as a manual transfer replication job, in a job sequence.

Execution of the job sequence is initiated by a controlling job that you define. The controlling job executes the `dm_run_dependent_jobs` method, which is installed with Content Server.

Repository implementation

A job sequence is stored in the repository as a set of `dm_job_sequence` objects. Each job sequence object represents one job in the sequence. The information in the job sequence attributes is used by the `dm_run_dependent_jobs` method (invoked by the controlling job) to execute the job and the sequence. For example, the information identifies the job to be executed, the job's predecessors, and the user as whom the job runs.

The value in the `object_name` attribute of job sequence objects is the name of the sequence. All job sequence objects representing one job sequence must have the same object name.

Job sequence execution

All jobs in a job sequence must be inactive. If a job in a sequence is an active job, it generates an error when the sequence is executed.

Execution of the sequence is initiated on the schedule you define for the sequence's controlling job. When that job is started, it invokes the `dm_run_dependent_jobs` method. The method controls the execution of the jobs in the sequence. The method initiates a job in the sequence by setting the job's `run_now` attribute to T. The job is then executed by the agent `exec` process the next time it polls the repository. The methods invoked by the sequenced jobs can be executed by the Java Method Server, the `dmbasic` method server, or Content Server.

The `dm_run_dependent_jobs` method begins by initiating all the jobs in the sequence that have no predecessors. When a job in the sequence completes, the method initiates any other jobs in the sequence whose predecessors have successfully completed. If a job

in the sequence fails, the method attempts to run the job up to two more times. If the job is not successfully run in three attempts, the method considers the job to have failed and records the information in the controlling job's `a_current_status` attribute. (You can view the current status information using Documentum Administrator.) Additionally, if a sequenced job fails, the `dm_run_dependent_jobs` method stops after allowing any remaining active jobs to complete. It does not initiate any more jobs.

The `dm_run_dependent_jobs` method finishes when there are no jobs in the sequence that are running and no jobs remaining to be run. The `dm_run_dependent_jobs` method returns one of two possible values on completion:

- 0, meaning every job in the sequence executed successfully
- 1, meaning some job or jobs in the sequence did not successfully complete, or rarely, that the method itself failed due to some problem after all jobs completed

Note: [Recovering from a job sequence failure, page 156](#), contains instructions on handling this situation.

Determining success for invoked jobs

The controlling method, `dm_run_dependent_jobs`, uses the values in the method attributes called `success_return_codes` and `success_status` to determine whether an invoked job completed successfully. For details of how this works, refer to [Defining success return codes and success status, page 142](#).

The repository connection file

The repository connection file is a file that contains connection information used by the `dm_run_dependent_jobs` method to connect to a repository to activate a job in the sequence. There is one repository connection file for each repository. The file is one or more individual lines that contain a server connection string, domain name, user name and an encrypted password.

The location of the file is provided to controlling jobs as a job argument. When `dm_run_dependent_jobs` wants to initiate a job, it searches the file for an entry that matches the values in the `job_docbase_name`, `job_login_user_name`, and `job_domain_name` attributes of the job's job sequence object. If a match is found, the method uses those values and the password in that entry to connect to the repository.

Using the file is not mandatory. If the argument identifying the file is not provided or if a match is not found, the method uses trusted login to connect to the repository. However, to use a trusted login, the server to which the method is connecting must reside on the same host as the controlling job. Additionally, the method must be running as the connecting user, and generally requires the connecting user to be the installation owner.

Typically, the file is created and maintained automatically by Documentum Administrator when you use Documentum Administrator to create and edit job sequences. Documentum Administrator creates the file and names it connect.dcf. It is stored in %DOCUMENTUM%\dba\config\repository_name\connect.dcf (\$DOCUMENTUM/dba/config/repository_name/connect.dcf). You can create and manage the file using the edit_dcf utility if you want to use another name or storage location. For instructions, refer to [Creating and maintaining a repository connection file for job sequences](#), page 153.

The agent exec process

All jobs are started by the agent exec process, a process that is installed with Content Server. At regular intervals, the agent exec process examines the job objects in the repository and runs those jobs that are ready for execution. (A job is ready for execution if the value in its a_next_invocation attribute is less than the current date and time or if its run_now attribute is set to T.)

There are several behavioral parameters that you can configure for the agent exec process, including how often it polls and how many jobs it executes each time. For instructions on configuring the agent exec, refer to [Modifying agent exec behavior](#), page 152.

Creating a job

You must have Sysadmin or Superuser privileges to create a job. Creating a job is easiest if you use Documentum Administrator. However, you can also use DQL statements. For instructions on using Documentum Administrator, refer to the Documentum Administrator online help.

Before you create a job, you may wish to read the information about scheduling jobs. Whether you decide to run a job as an individual job or as part of a sequence, you must set the job's schedule. Setting the schedule appropriately when the job is created will save steps after the job is created.

Similarly, you may wish to read the information about passing arguments before you create the job. [Passing arguments](#), page 149, describes how arguments are passed to the methods executed by jobs.

To create a job:

1. Write a Docbasic script, Java method, or other program to perform the required operations.
2. Create a method object that references the program created in [Step 1](#).

[Creating a method object, page 141](#), describes how to create a method object.

If the method is for a job that will be part of a job sequence, use the guidelines in [Defining success return codes and success status, page 142](#), to set the success return codes and success status fields.

3. Create a job object that points to the method.

If you intend to execute the job individually, activate the job.

If you intend to execute the job as part of a job sequence, inactivate the job.

Creating jobs in a multi-server environment

If you have multiple Content Servers running for the repository in which you are creating a job, you must identify a specific server for the job. That server, called the target server, is the repository server you want to execute the job. If you do not identify a specific server, multiple agent exec processes may attempt to start the job, which may result in version mismatch errors and erroneous reports of a failed job.

Using DQL to create a job

Use the DQL CREATE...OBJECT statement to create a job object. You must set the `method_name`, `start_date`, `run_mode`, `run_interval`, and `a_next_invocation` attributes. You will probably also want to set the optional attributes that set the job's name and expiration date, pass argument values to the method, and activate the job. (Refer to [Job, page 270](#), in the *EMC Documentum Object Reference Manual* for a description of job object attributes. The scheduling attributes are described in detail in [Scheduling jobs, page 148](#).)

Creating a job sequence

Before you create a job sequence, determine which jobs you want to include in the sequence and the order in which you want the jobs to execute. Use Documentum Administrator to create the sequence. You must have Sysadmin or Superuser privileges to create a job sequence.

Documentum Administrator only displays as choices for inclusion those jobs whose associated methods have a value for either or both of the following attributes: `success_return_codes` and `success_status` attributes. (For information about setting these attributes, refer to [Defining success return codes and success status, page 142](#).)

Use the following guidelines for the sequence:

- All included jobs must be set to inactive.
Documentum Administrator will allow you to add active jobs to a sequence. However, you must set such jobs to inactive before running the sequence.
- The controlling job must execute the `dm_run_dependent_jobs` method.
Documentum Administrator enforces this guideline and does not allow you to change the method when creating a job sequence.
- Set Pass Standard Arguments to true for the controlling job.

Creating the first job sequence automatically creates the repository connection file. Creating additional sequence modifies the file if needed. ([The repository connection file, page 145](#), describes the file and its purpose.)

Scheduling jobs

Jobs are executed on a defined schedule. If the job is executed as a standalone job (not in a job sequence), the execution schedule is controlled by attribute settings in the job. If the job is in a job sequence, its schedule is determined by a controlling job that activates the sequence. However, you must set the scheduling attributes of jobs in a sequence to valid values or the jobs fail when they are invoked in the sequence.

Defining a job schedule

Use Documentum Administrator to set job schedules. You must specify a starting date and the interval at which to execute the job. The interval is defined in the Repeat and Frequency fields.

Defining the starting date sets the following attributes:

- `start_date`
- `a_next_invocation`

The starting date is the earliest date at which the job can be executed. The `a_next_invocation` attribute defines the first (or next) scheduled execution of the job. By default, Documentum Administrator displays the current date and time as the default starting date. Reset this to the actual date and time on which you want the job to execute for the first time. After the job executes, the `a_next_invocation` attribute is reset to the date and time of the next invocation, computed using the interval you define.

The specific dates and times you set for a job (for example, an activation date or expiration date), are interpreted as server time. For example, suppose you set a job's start date to August 1 at 9 a.m. on a client machine in New York when the server machine is in The Hague. The tool becomes active when it is August 1, 9 a.m. in The Hague.

Defining the interval sets these underlying attributes:

- `run_mode`, set in the Repeat field
- `run_interval`, set in the Frequency field

These two values work in conjunction to define how often the job is run after its first execution. The Repeat field defines a unit of measure. The integer value you specify in the Frequency field is interpreted according to the unit of measure you select in the Repeat field.

For example, if you set the Repeat field (`run_mode`) to Hours and Frequency (`run_interval`) to 12, the job is executed every 12 hours. If you set Repeat to Week and Frequency to 7, the job is run every 7th day of the week.

In addition to specifying how often the job executes, you can specify how many times you want it to execute and when you want it to stop executing. The maximum number of executions is recorded in the `max_iterations` attribute. The end date for the job is recorded in the `expiration_date` attribute. Once either value is met, the agent exec process ignores the job and no longer schedules it for execution.

Passing arguments

Jobs have an attribute called `pass_standard_arguments` that determines which arguments are passed automatically to the method associated with the job.

If `pass_standard_arguments` is set to T, the server passes four standard arguments, listed in [Table 4-1, page 149](#), to the method. .

Table 4-1. Standard arguments passed to jobs

Argument	Datatype	Value	Description
<code>-docbase</code>	<code>string(64)</code>	<i>repository_name</i>	Identifies the repository
<code>-user_name</code>	<code>string(64)</code>	<i>user_name</i>	The user executing the job

Argument	Datatype	Value	Description
-job_id	ID	<i>job_object_id</i>	Object ID of the job
-method_trace_level	integer	<i>level</i>	Method trace level to use. The default is 0 (no tracing). The value for this argument is taken from the <code>method_trace_level</code> attribute of the job object.

If the server passes the standard arguments, it does not pass the method any arguments defined in the job's `method_arguments` attribute. The method must use the job ID passed as a standard argument to obtain the argument values defined in the job's `method_arguments` attribute.

If `pass_standard_arguments` is set to F, the server passes the method the argument values found in the job's `method_arguments` attribute, but does not pass the standard arguments.

The default for `pass_standard_arguments` is F.

Note: `pass_standard_arguments` is set to T for all the system administration jobs installed with Content Server. Do not reset that value to F.

The `run_now` attribute

The `run_now` attribute can be used for testing. If set to TRUE, the agent exec process runs the job the next time the process polls the repository for jobs ready for execution. The attribute does not affect the settings for `run_mode`, `run_interval`, or `a_next_invocation`. For example, if you modify a job's associated program, you can test it immediately by setting `run_now` to TRUE and saving the job. You do not have to wait to test the job until its next scheduled execution.

In Documentum Administrator, you set `run_now` by checking the Run After Update checkbox.

Managing jobs

This section contains information and procedures for managing jobs and job sequences. Included are the following topics:

- [Activating or inactivating a job, page 151](#)
- [Disabling all jobs, page 151](#)
- [Modifying agent exec behavior, page 152](#)
- [Creating and maintaining a repository connection file for job sequences, page 153](#)
- [Recovering from a job sequence failure, page 156](#)
- [Interpreting a job sequence status report, page 156](#)
- [Executing dm_run_dependent_jobs independently, page 157](#)

Activating or inactivating a job

When you set a job to inactive, the agent exec process ignores the job when it polls the repository and the job is never started by agent exec. When you set a job to active, the agent exec examines the job when it polls the repository and executes the job on the schedule defined in the job.

A job's activation status is recorded in its `is_inactive` attribute. To change the status, use Documentum Administrator.

A job can be set to inactive automatically if you have set a maximum number of executions for the job. In such cases, the job is inactivated after the specified number of executions are completed. Similarly, if you specify an expiration date for a job, it is inactivated on that date.

Disabling all jobs

To disable all job executions, set the `agent_launcher` attribute in the server config object to an empty string (""), and stop and restart the server. Stopping the server stops the current running agent exec process. When you restart the server, the `agent_exec` process is not launched.

Modifying agent exec behavior

The agent exec process controls the job execution. It runs continuously, polling the repository at intervals for jobs to execute. By default, only three jobs are allowed to execute in a polling cycle and tracing for the process is turned off.

You can change these default parameters, including the polling interval. The behavior is controlled by command line arguments in the `method_verb` argument of the method object that invokes the agent exec process. The arguments can appear in any order on the command line. You must have superuser privileges to change the `agent_exec_method`.

Setting the polling interval

The agent exec process runs continuously, polling the repository at specified intervals for jobs to execute. The default polling interval is controlled by the setting in the `database_refresh_interval` key in the `server.ini` file. By default, this is set to 1 minute (60 seconds).

To change the polling interval without affecting the database refresh interval, add the `-override_sleep_duration` argument with the desired value to the `agent_exec_method` command line. Use Documentum Administrator to add the argument to the command line. For example:

```
.\dm_agent_exec -override_sleep_duration 120
```

The polling interval value is expressed in seconds (120 is 2 minutes expressed as seconds). The minimum value is 1 second.

Setting the number of jobs in a polling cycle

By default, the agent exec executes up to three jobs in a polling cycle. To change the maximum number of jobs that can run in a polling cycle, add the `-max_concurrent_jobs` argument with the desired value to the `agent_exec_method` method command line. For example:

```
.\dm_agent_exec -max_concurrent_jobs 5
```

Use Documentum Administrator to modify the command line.

Turning on tracing for the agent exec process

Tracing for the agent exec process is turned off by default (trace level = 0). To turn it on, use Documentum Administrator to add the `-trace_level` argument to the `agent_exec_method` method command line. For example:

```
.\dm_agent_exec -trace_level 1
```

Setting the trace level to any value except zero turns on full tracing for the process.

The log file is named `agentexec.txt` and is stored in the `%DOCUMENTUM%\dba\log\repository_id\agentexec` (`$DOCUMENTUM/dba/log/repository_id/agentexec`) directory.

Creating and maintaining a repository connection file for job sequences

A repository connection file contains connection information used by the `dm_run_dependent_jobs` method to connect to the repositories that contain the jobs in the sequence. When the `dm_run_dependent_jobs` method executes, it searches the repository connection file to find an entry that specifies the server connection string, user, and domain identified in the job sequence object. If such an entry is found, it uses the password specified for that entry to make the connection to run the job.

Each entry in the file appears on a separate line and has the following format:

```
server_connect_string, [domain], user_name, password
```

Using this file is optional. If the `dm_run_dependent_jobs` method does not find a matching entry in the file or if the file doesn't exist, the method attempts to use a trusted login to invoke the sequenced job.

The file is typically maintained by Documentum Administrator when you edit or modify a job sequence. You can, however, use the `dcf_edit` utility to edit the file directly. Refer to [The `dcf_edit` utility, page 154](#), for instructions.

Specifying the server connect string

The `dm_run_dependent_jobs` method matches the value in the `server_connect_string` portion of the entry to the value in the job sequence object's `job_docbase_name` attribute when looking for a matching entry in the repository file.

The value can be any valid value used to designate a repository or server when connecting to a repository. For example, the following are valid formats for the values:

```
repository_name  
repository_name.content_server_name  
repository.content_server_name@host_name
```

The only requirement is that the value you define for the server connection string in the file must match the value specified in the `job_docbase_attribute` for the job in the sequence.

Commas and backslashes in the entries

You can use commas or backslashes in the values specified for the server connection string, domain, and user name by escaping them with a backslash. For example, “`doe\,john`” is interpreted as “`doe, john`”, and “`doe\\susie`” is interpreted as “`doe\susie`”.

You cannot use a backslash to escape any characters except commas and backslashes.

The `dcf_edit` utility

The `dcf_edit` utility allows you to add, remove, or replace entries in a repository connection file. It also allows you to backup the entire file. The utility is installed with Content Server as a method implemented using a Java class. The class is:

```
documentum.ecs.docbaseConnectionFile.DCFEdit
```

You can run the utility as a method from Documentum Administrator or as a Java command line utility. [Table 4-2, page 154](#), lists the arguments accepted by the method or on the command line.

Table 4-2. `dcf_edit` utility arguments

Argument	Description
<code>-server_config</code> <i>server_config_name</i>	Identifies the repository to which the utility will connect. This argument is required for Add and Remove operations, but is invalid for Backup operations.
<code>-login_domain</code> <i>domain_name</i>	The domain of the user identified in the <code>-login_user</code> argument This argument is optional for the Add and Remove operations, but is invalid for Backup operations.

Argument	Description
<code>-login_user <i>user_name</i></code>	Identifies the user as whom to connect. This argument is optional for the Add and Remove operations, but is invalid for Backup operations.
<code>-password <i>user_password</i></code>	The clear-text password for the user identified in <code>-login_user</code> . This argument is required for the Add operation, but is invalid for Remove and Backup operations.
<code>-f <i>repository_filepath</i></code>	Path to the repository connection file. This parameter is a required parameter for all operations.
<code>-operation <i>operation_name</i></code>	Identifies the operation being performed. Include only one operation on each execution of the utility. Valid operation names are: <ul style="list-style-type: none"> • add • remove • backup <p>Use add to add the connection information specified in the <code>server_config</code>, <code>user</code>, and <code>password</code> arguments to the file. If the entry already contains an entry with a matching server config value, this information replaces that entry. The password is encrypted before being added to the file. You must include the <code>-password</code> argument for an add operation. Including the <code>-login_user</code> argument is optional.</p> <p>Use remove to remove an entry from the file. The utility removes the entry with a value matching the <code>-server_config</code> name. Including the <code>-login_user</code> or <code>-password</code> arguments for a remove operation is optional.</p> <p>Use backup to create a backup of the file. The backup file is created in the same directory as the original file. The name is the file's name with an appended time stamp, in the format:</p> <p style="text-align: center;"><i>file_name-time_stamp</i></p> <p>Do not include the <code>-login_user</code> or <code>-password</code> arguments for backup operations.</p>

When the utility executes an Add operation, it looks for an entry in the file that matches the values you provide as arguments for `-server_config`, `-login_user`, and `-login_domain`. If a match is not found, the utility creates a new entry. If a match is found, the utility replaces the existing entry with the values in the arguments.

Recovering from a job sequence failure

If the controlling job exits with a status of 1, it typically means that one or more jobs in the sequence failed to complete successfully. To recover from that situation, take the following steps:

1. Examine the controlling job's status report to determine which jobs failed.
Use Documentum Administrator to view the job's status report. [Interpreting a job sequence status report, page 156](#), describes the entries in the report.
2. Examine the job reports of the failed jobs and the session and repository log files to determine the cause of the failure.
3. Correct the problem.
4. Run `dm_run_dependent_jobs` as a method, with the `-skip` argument, to re-execute the failed jobs.

[Executing `dm_run_dependent_jobs` independently, page 157](#), describes the arguments that you can include when you run the method independently of the job.

Interpreting a job sequence status report

The `dm_run_dependent_jobs` method, called by a job sequence's controlling job, generates a status report. You can view the status report using Documentum Administrator. [Table 4-3, page 156](#), lists the events recorded in the report and describes their format.

Table 4-3. Entries in a job sequence statusreport

Event	Entry Format
Start	<i>Date Time</i> start-sequence= <i>job_sequence_name</i>
Start Job	<i>Date Time</i> start job - docbase= <i>repository_name</i> job= <i>job_id</i> attempt=1 2 3

Event	Entry Format
End Job	<i>Date Time</i> end job - docbase= <i>repository_name</i> job= <i>job_id</i> result=succeed fail lastReturnCode= <i>a_</i> <i>last_return_code</i> lastDocumentID= <i>ID_of_job_report</i> lastJobStatus= <i>a_current_status</i>
End	<i>Date Time</i> end - job_sequence= <i>controlling_job_id</i> result=succeed fail
Error	<i>Date Time</i> error - docbase= <i>repository_name</i> job= <i>job_id</i> msg= <i>error_message</i>

Executing dm_run_dependent_jobs independently

You can execute the `dm_run_dependent_jobs` method independently of the controlling job. Use Documentum Administrator to run the method manually.

[Table 4-4, page 157](#), lists the arguments accepted by the method. Some arguments are required and some are optional.

Table 4-4. dm_run_dependent_jobs arguments

Argument	Required?	Description
<code>-docbase</code> <i>repository_name</i>	Yes	Identifies the repository that contains the job sequence.
<code>-user_name</code> <i>user_name</i>	Yes	Identifies the user executing the job sequence.
<code>-job_sequence</code> <i>sequence_name</i>	Yes	Identifies the job sequence to be executed.
<code>-method_trace_level</code> <i>trace_level</i>	No	Defines a trace level for the <code>dm_run_dependent_jobs</code> method. The two valid values are 0 and 10. 0 records status messages only. 10 provides debugging messages in addition to status messages. The default trace level is 0.

Argument	Required?	Description
-skip <i>list_of_jobs</i>	No	Identifies the jobs in the sequence that you do not want to execute. Provide a comma-separated list of job object IDs.
-dcf	See description	Specifies the location of the repository connection file. This is not required if the method is using trusted login to connect to the repositories in which the jobs in the sequence reside.

Clients and Sessions

This chapter contains information about Documentum clients and sessions. It begins with a definition and overview of each, and covers the following main topics:

- [Defining connection brokers for connection requests, page 164](#)
- [Failover and load balancing, page 166](#)
- [Bypassing the connection broker, page 166](#)
- [Requesting a specific server connection, page 167](#)
- [Requesting a native or secure connection, page 168](#)
- [Limiting which clients can access a repository, page 169](#)
- [Connection pooling, page 170](#)
- [Configuring login tickets, page 171](#)
- [Changing a session's configuration, page 172](#)
- [The dmcl.ini file, page 174](#)
- [Changing the assigned default operating system permissions \(UNIX only\), page 182](#)
- [Defining short date formats, page 182](#)
- [Disabling or enabling the client local area, page 183](#)
- [Changing the local area directory, page 188](#)
- [Monitoring local area disk space, page 190](#)
- [Removing content from local areas, page 191](#)
- [Managing persistent client caches, page 192](#)
- [Configuring DMCL exception handling, page 201](#)

Essential concepts

This section contains an overview of Documentum clients and sessions.

What is a client?

A *client* is an end user, application, or process that uses Content Server to access the repository.

A *client platform* is the machine on which the end user applications are running.

Documentum provides two basic client interfaces that allow end users to access repositories:

- Desktop (older versions are called Desktop Client)
- Webtop™

Desktop runs on Windows desktops. Webtop is a Web-based client. (For information about supported versions and platforms for either product, refer to its associated release notes.)

Client configuration

Just as you can set some operating parameters that are specific to repositories or servers, there are some operating parameters you can set which are specifically for the client platforms. For example, you can define a default short date format and specify which connection broker you want the client to use.

With the exception of the default short date format, the client's configuration parameters are defined in the client's dmcl.ini file. To change the client's configuration, you must change this file. The exception, the short date format, is defined individually on each client machine. (Refer to [Defining short date formats, page 182](#), for information about this procedure.)

The dmcl.ini file

The dmcl.ini file contains configuration and Content Server connection information for the clients. Each client must be able to access a dmcl.ini file. Each client must have a local copy of this file or be able to access a shared copy. Web clients, such as Webtop or Web Publisher, use the dmcl.ini file on the application server host.

When a client application opens an API session, some of the entries in the dmcl.ini file are read into the api config object. The api config object is used to determine the configuration of any subsequent repository sessions established during the API session. (Refer to [What is a session?, page 161](#), for a definition of an API session and a repository session. [The dmcl.ini file, page 174](#), describes the dmcl.ini's entries in detail.)

Note: Client applications talking directly to the server establish an api session to initialize the client library. If a user is using a Documentum client, this call is not necessary because the Documentum client issues the call when it is started up. For more information, refer to [Introducing repository sessions, page 29](#), in *Content Server Fundamentals*.

What is a session?

There are two types of sessions:

- API

An API session is started when an application instantiates the IDfClient class or issues the dmAPIInit call to establish communications with the Documentum API before it starts a repository session. (In the Documentum clients, this call is made for you automatically.) API sessions are not generally visible to end users, and they have no IDs. However, you can use the alias apisession or simply a in some method calls to refer to the API session. For example, you can use it in a Getdocbasemap method to obtain information about the repositories known to a connection broker.

API sessions are closed when the client application issues a dmAPIDeInit call or the application terminates.

- Repository

A repository session (hereafter called a session) is opened when an end user or application establishes a connection to a server. Each repository session has a unique session identifier, which is returned by the Connect method that establishes the server connection. The identifier has the format S_n where n is an integer equal to or greater than zero.

Repository sessions are only established within the context of an API session. That is, it is not possible to open a connection with a repository unless there is an open API session. During any single API session, an external application can have multiple repository sessions, with the same or different repositories.

A user or application's connection to the repository is terminated when the user or application issues a Disconnect method or when another user assumes ownership of the session. The session itself may also be terminated at that time or, if connection pooling is enabled, the connection may be held for future use in the connection pool.

User passwords are validated when a session first connects to a repository and are revalidated if the session times out and reconnects.

Inactive repository sessions are sessions in which the server connection has timed out but the client has not specifically disconnected from the server. If the client sends a request to the server, the inactive session automatically reestablishes its server connection and becomes active.

Session configuration

The behavior of any particular session, that is, its configuration, is determined by the values in its session config object.

Session config object

The session config object is a non-persistent object that is destroyed when the server connection is ended or the application terminates. It is constructed when the session is started using the values in the api config object, the server config object, and the connection config object. The session merges the attributes from each to build the session config object. (Refer to [Session Config, page 441](#), in the *EMC Documentum Object Reference Manual* for a complete description of the attributes of each object.)

Api config object

The api config object is a non-persistent object that is created when the client application issues a dmAPIInit call. Its attributes and their values reflect information found in the client dmcl.ini file. The api config object is destroyed when the application issues a dmAPIDeInit call.

There are a few attributes in the api config object that are also found in the server config object. When session is started, if the values for these attributes are not defined in the api config object, then the system uses their values in the server's corresponding attributes to build the session config object.

Server config object

The server config object is a persistent object stored in the repository. Every repository has at least one server config object that is created when the repository is created. The attributes in a server config object tell the server where to look for files, scripts, and executables that it needs.

Many of the attributes in the server config object are also found in the api config object. If the duplicated attributes are set in both objects, the system will take the values from the api config object for a session's session config object. Otherwise, it takes their values from the server config object. Refer to [Chapter 3, Servers](#), for more information about servers and server config objects.

Connection config object

A connection config object describes the configuration of a connection to a repository. The attributes in this object are derived from the server config object when the connection is established. If the connection is the first repository connection in the API session, it is also the primary session. In such cases, the attribute values in the connection config object are passed to the session config object constructed for the primary session.

Session connection attempts and timeouts

By default, when a client requests a connection to a server, the system attempts to connect and if it fails, the system makes another connection attempt immediately. If the second fails also, the system tries to connect to another server if another is available. If no other server is active, the client receives an error message.

Similarly, if a client session times out, the next time the user asks for a repository operation, Content Server re-authenticates the user and, if the authentication succeeds, the system automatically reconnects the user. Password validation is performed irrespective of whether a session has timed out or is connecting for the first time.

You can configure the time interval between connection attempts and the number of connection retries. To change these intervals, you set the `connect_retry_interval` and `connect_retry_limit` parameters, respectively, in the client's `dmcl.ini` file.

Session subconnections

If your enterprise has a multi-repository distributed configuration, users typically access both local and remote objects during a single repository session. An object is remote in a session if it does not reside in the repository to which that session is connected. When users access remote objects, the system creates subconnections under the primary session that connect to the repositories containing the remote objects.

Each subconnection has its own connection config object. The attributes in the object are derived from the server config object when the subconnection is established.

The maximum number of subconnections is set by the `max_connection_per_session` key in the `dmcl.ini` file. The default is 30 subconnections.

The maximum number of connections possible for a client process running on UNIX is limited by the descriptors set in the UNIX kernel. If a client process requires more connections than this limit permits, the UNIX system administrator must modify the UNIX kernel.

Viewing files

When a user requests to view a document, the server generally provides a copy of the document for the user. (The only exception is if the file is in a public storage area. In such cases, the user can view the file directly in the storage area.)

If the user is accessing the repository through Desktop, the server copies the document's content file to the user's client local area. If the user is accessing the repository through a Documentum Web client, such as Webtop, the server passes the document to the application server, which passes the document to the user's Web browser.

For Desktop, the default client local area is a directory called dmcl. The dmcl directory is created in the User Directory. The User Directory is identified when Desktop is installed and is typically c:\Documentum. (However, it may differ at your site.)

For Web clients, the application server sends the file directly to the user's Web browser for display to the user. The file is not copied to the application server's client local area.

Note: You can change the default for Desktop Client. For instructions, refer to [Changing the local area directory, page 188](#). You can change the default for Web clients, as well. For instructions, refer to documentation for the Web Development Kit and Webtop.

If you disable the use of the local area, the content file is copied to Content Server's common area. The common area is created during the server installation procedure. It is found at %DOCUMENTUM%\share\data\common (\$DOCUMENTUM/share/data/common).

For information about disabling local area use, refer to [Disabling or enabling the client local area, page 183](#).

Note: If you decide to use the server's common area, Documentum provides a secure writer program that you can use to ensure that files in this area are secure. (Files copied to the common area are owned by the server account by default. The secure writer changes permissions on the files so that they are owned by the user requesting the file and are inaccessible to other users.)

Defining connection brokers for connection requests

When a client requests a connection, the request is sent to a connection broker, which returns server connection information to the client. Which connection brokers can handle a client's connection request is defined in the client's dmcl.ini file. A dmcl.ini file specifies a client's primary connection broker and, optionally, up to 256 backup connection brokers. (For complete details about the dmcl.ini file, refer to [The dmcl.ini file, page 174](#).)

Note: The system is configured to use the backup connection broker entries, if present in the file and needed, by default. Refer to [Failover and load balancing, page 166](#), for more information.

Format of dmcl.ini sections

Sections for connection brokers in the dmcl.ini file have the following format:

```
[DOCBROKER_PRIMARY]
host = host_name      #required
port = integer         #optional
protocol = string      #currently unused

[DOCBROKER_BACKUP_n]
host = host_name
port = integer
protocol = string
```

For example:

```
[DOCBROKER_PRIMARY]
host = bigdog
port = 1489

[DOCBROKER_BACKUP_1]
host=lapcat
port=1489
```

The *host_name* is the name of the machine on which the connection broker resides.

The port is the TCP/IP port number that the connection broker is using for communications. The port number defaults to 1489 if it is not specified. If the connection broker is using a different port, you must include this key.

For example, the following entry in the dmcl.ini file defines bigcat as the host machine. Because the port is not specified, the client assumes the port is 1489:

```
[DOCBROKER_PRIMARY]
host=bigcat
```

In this next example, lapdog is defined as the primary connection broker. Because lapdog is not using the default port number, the port number is specified in the entry:

```
[DOCBROKER_PRIMARY]
host=lapdog
port=1491
```

The [DOCBROKER_PRIMARY] section is required. Including a backup section is optional; however, providing backup connection brokers can help ensure that all connection requests are successful. (If neither the primary nor any defined backup connection brokers respond, a connection error is reported to the user.)

If you specify backup connection brokers, add a separate [DOCBROKER_BACKUP_*n*] section for each backup. The *n* refers to the number of the backup. The backups are numbered sequentially, beginning with 0. For example, if you have two backups defined, there would be two backup entries: [DOCBROKER_BACKUP_0] and [DOCBROKER_BACKUP_1]. You can include up to 256 backup connection brokers .

In both primary and backup specifications, only the host key is required. The other keys are optional.

To add, change, or delete an section, use a text editor to edit the dmcl.ini file. The changes are visible immediately. You do not have to restart your session.

Failover and load balancing

Failover and load balancing for client requests to connection brokers is controlled by settings in the dmcl.ini file. The `auto_request_forward` key controls failover. The `docbroker_search_order` key controls load balancing. For information about setting up default failover, refer to [Failover for connection brokers, page 209](#). For information about setting up load balancing, refer to [Load balancing for connection brokers, page 209](#).

Bypassing the connection broker

There may be occasions when you need to bypass the connection brokers in order to debug some problem. To do this, add a [DOCBASE_DEBUG_BYPASS] section to the dmcl.ini file. This section has the following keys:

```
[DOCBROKER_DEBUG_BYPASS]
host = host_name
service = string
docbase_id = repository_id
port = port_number
```

The *host_name* is the name of host machine on which the server resides. The *service* is the server's TCP/IP service name as it would appear in a services file. The *docbase_id* is the ID of the repository the server accesses, expressed in decimal format. The *port_number* is the port number that the server is using for communications.

Important Note: Do not add this entry to a dmcl.ini file unless you are specifically going to connect with IAPI or IDQL to debug a problem. When this section is present, all connection requests are connected to the server specified in this entry, overriding any specified in the Connect method and bypassing the client's connection broker. Be sure to remove this entry when you have finished the debugging. Some Documentum clients cannot connect when this section is present.

Requesting a specific server connection

If you have started multiple servers for a repository, the Connect method syntax lets you be as specific as you like about which server you want to use for your repository connection. Depending on how you specify the *docbase* argument, you can:

- Let the DMCL choose the server for you

To allow the DMCL to choose which server your session uses, include only the repository name in the Connect request for the *docbase* argument. The connection broker will send the DMCL information about all servers for that repository and the DMCL will choose one.
- Choose a server by its name
- Choose any server residing on a specific host machine
- Choose a server by name that resides on a specific host machine

Requesting a server by name

To request a server by name, specify the repository argument as *docbase.server_name* in the Connect method. For example:

```
dmAPIGet ("connect, engrdocbase.bldg3server, johndoe, jdpasswr")
```

docbase.server_name can be a maximum of 120 characters.

If there is no server with the specified name that accesses the specified repository, the connection request fails.

To find out what servers are associated with a particular repository, use the *Getservermap* method. This method returns a server locator object that describes all the servers for a specified repository. (Refer to [Getservermap, page 256](#), in the *Content Server API Reference Manual* for details about using this method.)

Requesting a server on a specific host

You may want the connection broker to give you only connection information for a server that lives on a specific host machine. To do this, specify the *docbase* argument as *docbase@host_name* in the Connect method. For example:

```
dmAPIGet ("connect, engrdocbase@bldg3host, johndoe, jdpasswr")
```

docbase@host_name can be a maximum of 120 characters.

If there is no server on *bldg3host* that accesses the *engrdocbase*, the connection fails. If you want to ensure that you request a host that has at least one server that accesses

the requested repository, use the `Getservermap` method. This method returns a server locator object whose attributes describe all the servers known to the connection broker for a specified repository. One of the attributes, `r_host_name`, contains the name of the host on which the server resides. (Refer to [Getservermap](#), page 256, in the *Content Server API Reference Manual* for details about using `Getservermap`.)

Requesting a server by name on a specific host

You can tell the connection broker exactly which server you want on a specific host machine by specifying the `docbase` argument as `docbase.server_name@host_name` in the `Connect` method. For example:

```
dmAPIGet ("connect, engrdocbase.bldg3server@bldg3host, johndoe, jdpasswd")
```

`docbase.server_name@host_name` can be a maximum of 120 characters.

If the specified server does not reside on the specified host or if the server does not access the specified repository, the connection fails.

Requesting a native or secure connection

There are two types of connections: native and secure. A secure connection uses the secure socket layer protocol to provide a secure connection between Content Server and a client. A native connection is an unsecure connection. By default, all servers are configured to provide only native connections. If you installed the server with a trusted server license, you can configure the server to provide a secure connection. (For instructions, refer to [Setting the secure connection mode](#), page 109.)

If you are connecting to a server installed with a trusted server license, you can request a native or secure connection, depending on how the server is configured. The connection type requested by the client interacts with the server's configured connection type to determine whether the `Connect` command succeeds. For a description of the interaction, refer to [Table 2-41](#), page 161, in the *Content Server API Reference Manual*.

If you are connecting to a server that was not installed with a trusted server license, only a native connection is available.

There are two ways to specify which connection type you want:

- Set the `secure_connect_default` key in the `dmcl.ini` file
- Set the `secure_mode` argument in the `Connect` method

If you set the `secure_connect_default` key in a `dmcl.ini` file, the DMCL tries to start all client sessions that use that `dmcl.ini` with the connection type identified in the key. The `secure_connect_default` key defaults to native if you don't explicitly set it.

If you set the `secure_mode` argument in the `Connect` method, only the current session is started with the connection type identified in the argument. Setting the `Connect` argument overrides the connection type defined in the `dmcl.ini`. If the `secure_mode` argument is not included in the method, the DMCL uses the connection mode defined in the `dmcl.ini` file.

If you set neither the `dmcl.ini` key nor the `Connect` argument, the DMCL requests a native connection.

There are four values accepted by both the `dmcl.ini` key and the `Connect` argument. Those values are:

- `native`
 `native` means that you want only a native connection. If the DMCL cannot establish a native connection, the `Connect` method fails.
- `secure`
 `secure` means that you want a secure connection only. If the DMCL cannot establish a secure connection, the `Connect` fails.
- `try_secure_first`
 `try_secure_first` means that you prefer a secure connection, but will accept a native (unsecure) connection. The DMCL attempts to establish a secure connection first. If it cannot, it tries to establish a native connection. If that also fails, the `Connect` method fails.
- `try_native_first`
 `try_native_first` means that you prefer a native connection, but will accept a secure connection. The DMCL attempts to establish a native connection first. If it cannot, it tries to establish a secure connection. If that also fails, the `Connect` method fails.

Limiting which clients can access a repository

You can enforce which Documentum client versions may access a particular repository.

Two attributes of the `doibase` config object control client access: the `oldest_client_version` attribute and the `check_client_version` attribute. The `oldest_client_version` attribute is used by DFC to determine how to store chunked XML documents. Its value must be set manually.

When `check_client_version` is set to `TRUE`, then the value of `oldest_client_version` is also used to determine the oldest Documentum client version that may access the repository.

If Retention Policy Services or Collaborative Services is enabled in the repository, the value of `oldest_client_version` is set to 5.3 and the value of `check_client_version` is set to `TRUE`. If older clients must access that repository, you must manually set

check_client_version to FALSE. If Retention Policy Services is enabled, the older clients can bypass the restrictions imposed by Retention Policy Services.

Connection pooling

Connection pooling allows applications to reuse sessions. When connection pooling is enabled, the client library creates a connection pool when an API session is started. Each new repository session is registered with the connection pool. When a user disconnects from a session, the session is held in the pool to be used the next time the user or a different user requests a connection to the repository. How many times any particular connection can be re-used is configurable.

Using connection pooling provides performance benefits for applications that execute frequent connects and disconnects with repositories. For example, a web-based client application that services many users can make use of connection pooling to provide optimal performance for users.

Enabling connection pooling

To enable connection pooling, set the connect_pooling_enabled key in the client's dmcl.ini file to TRUE.

Setting the key affects all sessions that are opened after the key is set. Sessions that are open when the key is set are not affected. Any applications that are running in the current API or repository sessions are not affected either. They must be restarted if you want them to use connection pooling.

Setting the session recycle limit

A limit is imposed on how long a session in the connection pool can exist. When the limit is reached, the session is destroyed and a new session is created in the connection pool. This allows various caches to be cleaned up and stale session data to be refreshed regularly. The limit is defined in the connect_recycle_interval key in the dmcl.ini file.

When a session is started, its creation time is noted and each time the session is disconnected, the time at disconnection is compared to the creation time. If the difference between the two times is greater than the interval specified in connect_recycle_interval, the session is destroyed and a new session is created to replace the destroyed session.

By default , the `connect_recycle_interval` key is set to 300 seconds, which is 5 minutes. To reset this key, use Documentum Administrator.

Removing free connections from the pPool

There may be occasions when you want to manually remove free connections from the connection pool. To do that, use the `Flushconnectpool` method. The method disconnects all free connections and removes them from the connection pool. The method affects only the connection pool used by the application that issues the method. Any user can issue this method. The method is described in [Flushconnectpool, page 219](#), in the *Content Server API Reference Manual*.

Configuring login tickets

Login tickets are values that applications can use in place of a user's password in a DMCL method that establishes a repository session or authenticates a user. Login tickets are obtained by executing the `Getlogin` method.

You can configure the length of time for which a ticket is valid and, indirectly, the number of tickets that a Content Server can maintain concurrently.

Setting ticket validity period

To define the length of time for which a login ticket is valid, set the `login_ticket_timeout` attribute in the server config object. By default, the attribute is set to 5 minutes.

After you set the attribute, you must reinitialize the server to make the change take effect.

Setting the ticket cache size for Content Server

By default, the maximum number of tickets with server scope that can be cached in Content Server is equal to 10 times the maximum number of concurrent sessions. For example, if the maximum number of concurrent sessions allowed is 1000, then the maximum number of tickets that can be cached is 10,000.

You can change the multiplier (10) by setting the `ticket_multiplier` key in the `server.ini` file.

The number of concurrent sessions is controlled by the `concurrent_sessions` key in the `server.ini` file.

Changing a session's configuration

A session's configuration, or operating behavior, is determined by the values of the attributes in its session config object. The session config object is constructed when the session is started, using the attributes of the api config object and the server's server config object.

You can change the behavior of your current session, any subsequent session, or both, depending on which config object you change and whether you change the `dmcl.ini` file. [The scope of changes, page 172](#), describes how changes you make to the client or session's configuration influence current and future sessions.

General procedure

You can change almost all the attributes that govern session behavior. For example, you can set the `client_cache_size` or you could change the number of rows returned to the server for each call to the RDBMS.

To change a session's behavior, you set or modify the attribute associated with the behavior you want to change and then use the `Restart` or `Reinit` method to reinitialize the server.

Some attributes are defined for more than one object. For example, `nfs_enabled` can be reset in the session config object, in the api config object, or in the `dmcl.ini` file. Which object you change and which method you use to reinitialize the server will determine the scope of your changes.

This chapter provides detailed instructions for only two common changes—disabling/re-enabling the use of the client local area and changing the location of the client local area. Refer to a description of the session config, api config, and server config object types in [Chapter 2, Object Reference](#), of the *EMC Documentum Object Reference Manual* for a complete list of the attributes of each.

The scope of changes

The scope, or range of influence, of any changes you make to the session or client configurations depends on which object you change and whether you use `Reinit` or

Restart to reinitialize the server. For example, if you change the session config object, then only your current repository session is affected. [Table 5-1, page 173](#), describes the objects you can change and the scope of each change.

Table 5-1. Scope of changes to client and session configuration parameters

Changing this:	Affects:
session config object	Current session.
api config object	Sessions in the same API session but subsequent to current session. If you restart the server, this also affects the current session.
dmcl.ini file	Future API sessions and all repository sessions occurring within them.
server config object	If you reinitialize the server (using the Reinit method) after you change and save the server config object, the changes affect your current session and any subsequent sessions. Note that current sessions other than yours are not affected by the changes. If you restart the server (using the Restart method) after you change and save the server config object, the changes affect only your current session. (Future sessions are not affected until you reinitialize the server.) Restarting the server is a useful way to test changes before making them visible to other sessions.

The Restart and Reinit methods

These methods are both used to reinitialize a server. They differ in the scope of their effect. The Restart method reinitializes only the session thread (spawned server process) to which your current session is connected. The Reinit method reinitializes the main server thread (parent server process) and the session thread (spawned server process). Neither method affects the session threads (spawned servers) for other current sessions with the repository.

The Restart method has an optional flag that allows you to reinitialize only the client portion of a session's configuration. With this flag, you can change the local area

directory or turn off its use and reinitialize the server without affecting the server portion of the configuration.

The dmcl.ini file

The dmcl.ini file defines a client's configuration. This file has one required section and five optional sections. Documentum provides a file, dmclfull.ini, that describes all the possible sections and search rules for a dmcl.ini file. The dmclfull.ini file is found in %DOCUMENTUM%\product\version\bin (\$DOCUMENTUM/product/version/bin).

Adding keys to the file

On Windows, the names of the keys are not case sensitive. On UNIX, the names of the keys are case sensitive.

Adding comments to the file

If you want to add a comment to the file, use a semi-colon (;) as the comment character.

The DMCL_MASTER section

This optional section defines the location of a master dmcl.ini file. If it is present in a client's dmcl.ini file, the client reads the values in the master file first and then reads the values in the local file. If there are keys that are set in both files, the data in the local file overrides any settings in the master file.

The DMCL_MASTER section has only one key:

```
[DMCL_MASTER]    #Optional entry  
path = string
```

string must be a path specification for the directory that contains the master dmcl.ini file.

The DMAPI_CONFIGURATION section

The values of the keys in this section are copied into the api config object when it is constructed following a dmAPIInit call. If any value is not set, or the entire section is not

present, either a default value or the value from the corresponding attribute in the server config object is used. [Table 5-2, page 175](#), lists the keys in the DMAPI_CONFIGURATION section. Refer to [Api Config, page 68](#), in the *EMC Documentum Object Reference Manual* for an explanation of the attributes of the api config object.

Table 5-2. dmcl.ini DMAPI_CONFIGURATION keys

Key	Datatype	Comments
auto_request_forward	Boolean	Default is T. For information about its use, refer to Failover and load balancing, page 166 .
batch_hint_size	integer	Defines the number of rows returned to Content Server by the RDBMS in a single call to the RDBMS. Default is 20.
cache_queries	Boolean	Default is F. If set to T and client_persistent_caching is T, then query results obtained by a Cachequery method are cached.
castore_write_max_attempts	Integer	Defines the maximum number of write attempts when writing content to a content-addressed storage area. The default is 3.
client_cache_size	integer	If explicitly set, the value is used to define the maximum size of the client cache. If this key is defaulted, the DMCL uses the value in the corresponding key in the server config object to determine a session's maximum client cache size. The value is interpreted as the number of objects. The maximum size can be exceeded if a new object must be added to the cache and no candidates for deletion are found. Also, persistently cached objects do not count towards the maximum. Default is -1.
client_cache_write_interval	integer	Defines how often periodic updates of the persistent cache on the local

Key	Datatype	Comments
		disk is updated during a session. Refer to Defining the persistent cache write interval, page 198 for more information.
client_codepage	string	The value is dependent on the language and OS codepage of the host machine.
client_locale	string	The value is dependent on the language and OS codepage of the host machine.
client_os_codepage	string	The value is dependent on the language and OS codepage of the host machine.
client_persistent_caching	Boolean	Controls whether client persistent caching is enabled. The default is T, caching is enabled. Refer to Enabling and disabling persistent client caching, page 192 for more information about this key.
connect_pooling_enabled	Boolean	Default is F. For information about this key, refer to Enabling connection pooling, page 170
connect_recycle_interval	integer	Default is 300. The value is expressed as seconds. For more information about this key, refer to Setting the session recycle limit, page 170
connect_retry_interval	integer	Default is 0 (no wait).
connect_retry_limit	integer	Default is 2.
docbroker_search_order	string	Default is sequential. The only other valid value is random. For more information, refer to Failover and load balancing, page 166 .

Key	Datatype	Comments
exception_count	integer	This key is supported on Windows platforms only.
exception_count_interval	integer	<p>Defines how many exceptions the DMCL records. The default is 0.</p> <p>This key is supported on Windows platforms only.</p> <p>Defines a time interval in which the number of recorded exceptions may not exceed the value in exception_count. If the number of recorded exceptions exceeds the value of exception_count with the given interval, the DMCL shuts down.</p> <p>The value is interpreted in minutes.</p> <p>The default is 0.</p>
extern_store_content_static	Boolean	<p>Controls how Getfile methods behave within a session when fetching content from external storage areas. For an explanation of the behavior, refer to Configuring for optimal performance on retrieval, page 255.</p> <p>The default is F (FALSE).</p>
force_coherency_checks	Boolean	<p>Setting this to T (TRUE) disables the persistent caching consistency check rules in the client. The default is F.</p> <p>For more information, refer to Overriding consistency checking rules, page 198.</p>
local_diskfull_limit	integer	<p>Default is 100.</p> <p>For information about this key, refer to Monitoring local area disk space, page 190</p>

Key	Datatype	Comments
local_diskfull_warn	integer	Default is 100. For information about this key, refer to Monitoring local area disk space, page 190
local_path	string	Identifies the directory location of the persistent client caches (and where content files are written if no directory is specified in the Getfile). The default is the current working directory.
local_purge_on_diskfull	Boolean	Default is T. For information about this key, refer to Monitoring local area disk space, page 190
max_collection_count	integer	Default is 10.
max_connection_per_session	integer	Default is 30.
max_session_count	integer	Default is 10.
network_callback_enabled	Boolean	Default is T.
nfs_enabled	Boolean	Default is F. For information about this key, refer to Disabling or enabling the client local area, page 183
persistent_cache_write_interval	integer	Used to manage persistent client caching. Refer to Defining the persistent cache write interval, page 198 for details.
secure_connect_default	string	Default is native. For information about this key, refer to Requesting a native or secure connection, page 168

Key	Datatype	Comments
terminate_on_exception	Boolean	<p>This is supported only on Windows clients.</p> <p>Controls whether the DMCL shuts down on exception. T (TRUE) means that the process is allowed to terminate. F (FALSE) means that the process is allowed to continue but the DMCL calls will fail with an error.</p> <p>The default is T.</p>
token_storage_enabled	Boolean	<p>Controls whether the client library can retrieve application access control tokens from the location specified in <code>token_storage_path</code>, for use in connection requests.</p> <p>Refer to Enabling token retrieval by the client library, page 452, for information about how the defaults are set.</p>
token_storage_path	string	<p>Directory location of the token files generated by the <code>dmtkgen</code> utility.</p> <p>The default setting is:</p> <pre>E:\Documentum\apptoken</pre> <p>For information about resetting this key, refer to Enabling token retrieval by the client library, page 452</p>
trace_file	string	Default is the server log.
trace_level	integer	Default is 3.
umask	string(4)	This is supported on UNIX platforms only. Refer to Changing the assigned default operating system permissions (UNIX only), page 182 for information about its use.
use_compression	Boolean	Default is F.

Key	Datatype	Comments
use_content_server	Boolean	Default is T. For information about this key, refer to Using the use_content_server key, page 113
use_local_always	Boolean	Default is F.
use_local_on_copy	Boolean	Default is F.

The DOCBROKER sections

The dmcl.ini file also contains sections that define the connection brokers to which the client sends all of its connection requests. For information about these sections, refer to [Defining connection brokers for connection requests, page 164](#).

The DOCBASE_OVERRIDE _docbase section

This section contains keys whose values override the drive letter aliases defined in the mount point object in your repository that represents the mounted %DOCUMENTUM%\share (\$DOCUMENTUM/share) directory. Sometimes, a group of users may already have mounted another directory using the drive letter alias specified in the mount point object for the share directory. In those instances, you need a way to force the system to use a different alias than that specified in the mount point object.

The DOCBASE_OVERRIDE_docbase section provides that capability. You can give that set of users a separate copy of the dmcl.ini file that includes a DOCBASE_OVERRIDE_docbase section that defines alternate drive letter aliases for three platforms. This section has the following keys:

```
[DOCBASE_OVERRIDE_docbase]
common_mtpt_windows_alias = string
common_mtpt_macintosh_alias = string
common_mtpt_unix_alias = string
```

The common_mtpt_windows_alias key defines an alternate drive alias for Windows clients; the common_mtpt_macintosh_alias defines an alternate drive alias for Macintosh machines; and the common_mtpt_unix_alias defines an alternate drive alias for Motif machines.

Because each repository has a mount point object defined for the %DOCUMENTUM%\share (\$DOCUMENTUM/share) directory, you should put one

DOCBASE_OVERRIDE_*docbase* section in the dmcl.ini file for each repository. Specify the name of the repository in the section name, for example:

```
[DOCBASE_OVERRIDE_engrdb]
. . .
```

Finding the dmcl.ini file

The location of the dmcl.ini file is dependent on the Documentum client and the client's host machine.

Content Server installations

When you install Content Server on a UNIX or Linux host, the procedure creates a dmcl.ini file in \$DOCUMENTUM. On Windows hosts, the dmcl.ini file is created in C:\WINNT. This dmcl.ini file is used by internal jobs and methods, such as dm_DistOperations, that must connect to a repository.

Desktop

The dmcl.ini file is found in the Windows directory.

WDK applications

If the application server is installed on a Windows host, when the WDK application is installed, the dmcl.ini file is created in C:\WINNT on the application server host. If the application server is installed on a UNIX or Linux host, the dmcl.ini file is created in \$DOCUMENTUM.

Changing the dmcl.ini file

To change the dmcl.ini file, use a text editor.

If users are working in a Documentum client, changes to the dmcl.ini file are not visible until they close their current client session and open another. For external client applications, the changes are not visible until the application closes its connection to the API (with a dmAPIDeInit call) and opens another (with a dmAPIInit call).

Changing the assigned default operating system permissions (UNIX only)

When the client DMCL creates directories on the client machines or writes files to directories on the client machines, it assigns default operating system permissions to those directories and files. The default permissions assigned to directories are 777 and the default permissions assigned to files are 666. You can change the defaults assigned to public directories and files by setting the `umask` key in the `dmcl.ini` file or by setting the `umask` attribute in the `api` config object.

Setting `umask` in the `dmcl.ini` file affects all public directories and files created during sessions established using that `dmcl.ini` file. Setting it in the `api` config object affects only directories and files created during repository sessions within that `api` session.

The `umask` value works similarly to the UNIX `umask` functionality. The value is subtracted from the default permissions to determine the actual permissions assigned to a file or directory. For example, if you set `umask=2`, then the default permissions assigned to directories becomes 775 and the default permissions for files becomes 662. Or, if you set `umask=20`, then the permissions become 757 for directories and 626 for files.

Defining short date formats

Different parts of the world have different short-hand ways to write a date. For example, some people use `mm/dd/yy` and others use `dd-mm-yyyy`. To accommodate these differences, the client platforms allow users to specify a short date format. The server recognizes all of these user-specified short date formats as valid formats for date input.

If a short date format is defined for a client machine, the server displays dates on that client using the short date format. (For more information about how date formats are used, refer to [Default formats, page 16](#), of the *Content Server DQL Reference Manual*.) If no short date format is defined on a client machine, the server uses `mm/dd/yyyy` as the default.

On Windows clients, use Control Panel>Regional Settings>Date tab to set the short date format.

On UNIX clients, defining the short date differs by machine:

- For SunOS and HP-UX clients, there is no way to define a short date format. The server assumes the format in these cases to be `mm/dd/yy hh:mm:ss`.
- For Solaris and AIX, the short date format is the format defined by the machine's locale. (Locale is set by the `setlocale` command.) In most cases, this is already set to the appropriate value. However, if it is not, you can set the time and date to use the appropriate locale with the following command:

```
setenv LC_TIME local_code
```

- Refer to your UNIX System Administrator's manual for the Solaris machine for the valid local codes.

Disabling or enabling the client local area

In most cases, when a user asks to view a document, the server makes a copy of the document's content file for the user to view. (The exception can occur when the content file is stored in a public storage area. In those cases, the user may be allowed to view the content file in the storage area.)

The server can put the document copy in either the user's client local area (a directory on the user's local disk) or the server's common area (a directory on the server's host machine). In a default configuration, the server uses the client local area.

This configuration is controlled by one attribute in the server config object and three keys in the dmcl.ini file. The server config attribute is `nfs_enabled`. The dmcl.ini keys are:

- `nfs_enabled`
- `use_local_always`
- `use_local_on_copy`

The attribute and all three keys are set to FALSE by default. If you set the `nfs_enabled` attribute and the `nfs_enabled` key to TRUE and don't change the `use_local_always` or `use_local_on_copy` keys, the server will copy files for viewing to the server common area instead of the user's local area. If you change either `use_local_always` or `use_local_on_copy` to TRUE, the server will continue to use the client local area in certain circumstances:

- If the `use_local_always` key is set to TRUE, the server will always copy files for viewing to the user's local disk even if the `nfs_enabled` attribute and key are set to TRUE.
- If you set `use_local_on_copy` key to TRUE, it directs the server to copy files for viewing to the local area whenever the user has not specified a location and file name for the requested file.
- The setting for `use_local_on_copy` is only used by the server if `use_local_always` is FALSE and `nfs_enabled` is TRUE. In these cases, it provides a way to direct the server to use the client local area when it is most advantageous but allows the server to continue to give clients viewing access to files in public storage without copying them.

You cannot set both `use_local_always` and `use_local_on_copy` to TRUE. Only one or the other can be TRUE.

[Table 5-3, page 184](#), summarizes the possible settings and the behavior of the server in each case.

Table 5-3. Configuration settings for uUse of client local area and server common area

Settings				Server behavior
nfs_enabled attribute	nfs_enabled key	use local always	use local on copy	
F	F	T or F	T or F	Default. The server copies files for viewing to the client's local area.
T	F	T or F	T or F	The server copies files for viewing to the client's local area.
F	T	T or F	T or F	The server copies files for viewing to the client's local area.
T	T	F	F	The server always copies files for viewing to the server common area.
T	T	T	F	The server always copies files for viewing to the client's local area.
T	T	F	T	The server copies files for viewing to the client's local area whenever the user doesn't specify a location and

Settings				Server behavior
nfs_enabled attribute	nfs_enabled key	use local always	use local on copy	file name for the copied file.

Scope of the change

Disabling the use of the local area requires you to reset the `nfs_enabled` attribute and, sometimes, the `use_local_always` or `use_local_on_copy` attribute. Because these attributes appear in more than one object, which sessions are affected when you disable (or re-enable) use of the client local area depends on which objects you change to achieve the effect. For example, if you set the `nfs_enabled` to `TRUE` and `use_local_always` attribute to `FALSE` in the session config object, then your current session, and only your current session, is affected.

[Table 5-1, page 173](#), summarizes how changing session config objects, api config objects, the `dmcl.ini` file, and server config objects affects your current and future sessions.

This section describes how to change your settings to affect your current repository session and how to change them for all future api or repository sessions.

Disabling local area use

In the basic installation, using the local area is enabled by default. This section provides procedures for disabling it use.

If you disable the use of the local area, the client must have access to the server's common area. Be sure that capability is functioning before you disable the use of the client's local area.

In the current repository session

Within an API session, you can have multiple repository sessions, some of which may be open simultaneously. This procedure only disables the local area for your current session; it does not affect any other users who may have sessions open with the same repository or server.

Use the following steps to disable local area use for the current repository session:

1. Use the Get method to check the values of the `nfs_enabled`, `use_local_always`, and `use_local_on_copy` attributes in the session config object:

```
get, session, sessionconfig, nfs_enabled
get, session, sessionconfig, use_local_always
get, session, sessionconfig, use_local_on_copy
```

2. Use the Set method to change these values, as necessary, to FALSE for `use_local_always` and `use_local_on_copy` and TRUE for `nfs_enabled`:

```
set, session, sessionconfig, use_local_always, FALSE
set, session, sessionconfig, use_local_on_copy, FALSE
set, session, sessionconfig, nfs_enabled, TRUE
```

3. Execute the Restart method to reinitialize your current session's server.

```
restart, session
```

For all repository sessions

This procedure disables the use of the client local area for all future repository sessions, whether they occur in your current API session or in future API sessions. The procedure doesn't affect users who have repository sessions currently open with the server.

To disable the client local area for all repository sessions:

1. Fetch the server's server config object.
2. Set the server's server config object's `nfs_enabled` attribute to TRUE.
3. Save the server config object.
4. Edit the `dmcl.ini` file to ensure that the `use_local_always` and `use_local_on_copy` keys, if present, are set to FALSE.
5. Make sure the `use_local_always` and `use_local_on_copy` attributes in the api config object are set to FALSE.

You can use the Get method to obtain the values in these attributes:

```
get, session, apiconfig, use_local_always
get, session, apiconfig, use_local_on_copy
```

If they are not FALSE, use the Set method to set them to FALSE:

```
set, session, apiconfig, use_local_always, FALSE
set, session, apiconfig, use_local_on_copy, FALSE
```

6. Issue the Reinit method to reinitialize the parent server.

```
reinit, session
```

Enabling local area use

This section tells you how to turn the use of local area back on if you have disabled it. The procedures tell you how to re-enable it in your current repository session and for all future sessions.

In the current repository session

Within an API session, you can have multiple repository sessions, some of which may be open simultaneously. This procedure describes how to re-enable the use of the local area in your current repository session. Other users who have a current session open with the server will not be affected by this procedure.

Use the following steps to re-enable local area use for the current repository session:

1. Use the Get method to check the value of the `nfs_enabled` attribute in the session config object:

```
get,session,sessionconfig,nfs_enabled
```

2. If `nfs_enabled` is TRUE, to re-enable the client local area, you can:

- Use the Set method to change `nfs_enabled` to FALSE:

```
set,session,sessionconfig,nfs_enabled,FALSE
```

or

- Use the Set method to set `use_local_always` to TRUE:

```
set,session,sessionconfig,use_local_always,TRUE
```

You cannot set `use_local_always` if `use_local_on_copy` is set to true.

or

- Use the Set method to set `use_local_on_copy` to TRUE:

```
set,session,sessionconfig,use_local_on_copy,TRUE
```

You cannot set `use_local_on_copy` if `use_local_always` is set to true.

(Refer to the beginning of [Disabling local area use, page 185](#), and [Enabling local area use, page 187](#), for information about the differences between `use_local_on_always` and `use_local_on_copy`.)

3. Execute the Restart method to reinitialize your current session's server.

```
restart,session
```

In all future repository sessions

This procedure re-enables the client local area for your current repository session and any future sessions. It does not affect any users who have a session currently open with the server.

To re-enable client local use for the current and future repository sessions:

1. Fetch the server's server config object.
2. Set the server's server config object's `nfs_enabled` attribute to `FALSE`.
3. Save the server config object.
4. Issue the `Reinit` method to reinitialize the parent server.

```
reinit, session
```

Changing the local area directory

You can change the directory location of the client local area. (Refer to [Viewing files](#), page 164, for information about the default location of the client local area.)

You can define a new location for the duration of the current repository session, for all repository sessions in the current API session, or for all future API and repository sessions.

In the current API session

Within an API session, you can have multiple repository sessions, some of which may be open simultaneously. You can change the local area for only your current repository session, or for all future sessions, or for your current session and all future sessions.

For the current repository session

To change the local area in use for the current repository session:

1. Set the `local_path` attribute in the session config object to the new directory.
The directory must exist before you do this, and its permissions must allow the world to read and write it.
2. Issue the `Restart` method with the `thereset_client` flag set to `TRUE` to make the change take effect:

```
restart,session,,T
```

The extra comma in the syntax is a place holder for the name of the server's server config object, an optional argument that is not specified in this example. You can specify it if you like:

```
restart,session,server_config_name,T
```

The *reset_client* flag tells the system to reinitialize only the server parameters that define the client's configuration. The reinitialization does not affect any parameters, such as the number of concurrent users, that only affect the server's behavior. The flag is FALSE by default.

To affect future repository sessions

To change the local area for future repository sessions in the current api session, set the *local_path* attribute in the api config object to the new directory.

```
set,session,apiconfig,local_path,new_directory
```

To Affect Current and Future Repository Sessions

To change the local area for your current session and subsequent repository sessions in the current API session:

1. Set the *local_path* attribute in the api config object to the new directory.

```
set,session,apiconfig,local_path,new_directory
```

2. Issue the Restart method to reinitialize your current session's server.

```
restart,session,,reset_clients
```

The extra comma in the syntax is a place holder for the name of the server's server config object, an optional argument that is not specified in this example. You can specify it if you like:

```
restart,session,server_config_name,T
```

The *reset_client* flag tells the system to reinitialize only the server parameters that define the client's configuration. The reinitialization does not affect any parameters, such as the number of concurrent users, that only affect the server's behavior. The flag is FALSE by default.

In all future repository sessions

To change the local area for any future repository session in any API session:

1. Edit the dmcl.ini file to set the local_path key in the file to the new directory.
2. Set the local_path attribute in the api config object to the new directory.
`set, session, apiconfig, local_path, new_directory_name`
3. Use the Reinit method to reinitialize the server:

```
reinit, session
```

Because you set the api config object and reinitialized the server, you will see your changes in your current session. Other users will see the changes as soon as they open a repository session. Changes, for them, are not visible in their current sessions.

Monitoring local area disk space

There are two keys in the dmcl.ini file that let you define boundaries for disk space usage for the client local areas. These are:

- local_diskfull_limit

This is an integer key. It determines how full you want to allow the disk containing the local area to become. You can assign values between 1 and 100. The value represents a percentage of the total possible space on the disk. If copying a file into the local area will fill the space beyond this limit, the server generates an error and does not copy the file.

The default setting is 100.

- local_diskfull_warn

This is an integer key. Like the local_diskfull_limit, valid values range from 1 to 100 and represent a percentage of the total possible space on the disk. If copying a file into the local area will fill the space beyond this percentage, the server sends a warning to the client. The file is copied to the local area, however.

The default setting is 100.

Note: The errors and warning generated by local_diskfull_limit and local_diskfull_warn are not displayed directly to the client. The application must check for these after every Getfile to the local area.

By default, when a client reaches the local_diskfull_limit, the server automatically removes all the content files from the client's local area that were fetched during that client's session. This behavior is controlled by the local_purge_on_diskfull key, defined in the dmcl.ini file. It is a Boolean key and is set to TRUE by default during the installation procedure.

When an API session starts, the key's value is copied into the api config object, and from there, into the session config object when a repository session is started. Where you choose to change the value will depend on what scope you want the change to have. For example, if you change its value in the session config object, it affects only your current repository session. If you change it in the dmcl.ini file, it affects all future sessions. Refer to [Changing a session's configuration, page 172](#), for details about changes and their scope.

Removing content from local areas

The system automatically removes from the local area the files that users requested during their sessions when they terminate the sessions. If a session terminates abnormally, for example, if the machine crashes, the local area is not cleaned up. However, each time a session is started on a client machine, the system scans the machine's local area and cleans up any files not being used by an active session. This means that the files left behind by an abnormal session termination are removed when the next session is started.

The system uses the session ID in the file's path to determine if the file is in use or not. If the session is an active session, the file is not removed during the clean up. If the session is not an active session, the file is removed.

Note: *Note:* Files copied to client local areas are given names in the following format:

```
\root\docbase_id\machine_name\session_id\file_name
```

or, on UNIX:

```
/root/docbase_id/machine_name/session_id/file_name
```

Manual clean up

If you receive an error or warning because you have exceeded the limits specified in `local_diskfull_limit` or `local_diskfull_warn`, you can use the `Purgelocal` method to remove the files in your local area.

The `Purgelocal` method removes from the local area copies of any files that you have requested during your current session. It does not remove files copied into the area by other active sessions. For example, if you have two sessions, A and B, open, executing a `Purgelocal` from session A only removes files copied into the area during session A. The files requested by session B are not touched.

The syntax of `Purgelocal` is:

```
dmAPIExec(purgelocal, session)
```

Any user can use this method.

Managing persistent client caches

Persistent client caches are caches of objects and query results managed by the client DMCL and maintained across sessions. Objects and query results are placed in the caches by specific request when an application or user issues a Fetch or a Query_cmd method. (The Cachequery method also caches query results, but Query_cmd is the recommended method to use for caching query results.) Typically, the cached objects and query results are objects and results that change very infrequently. If objects or results that change frequently are cached, the cached copies must be updated regularly and the benefits of caching are diminished.

Persistent client caching is enabled by default in repositories and client sessions. Documentum Webtop and Documentum Desktop take advantage of this feature automatically. User applications can also take advantage of persistent client caching by using the API methods that support object and query caching.

Cached data is checked for consistency with the repository based on consistency check rules defined in the Fetch and Query_cmd methods that reference the data. Objects and query results can be consistency checked individually or a set of data can be defined in a cache config object and checked as a set.

For a complete description of the feature, refer to [Persistent client caches, page 45](#), in *Content Server Fundamentals*. [Using persistent client caching in an application, page 48](#), in *Content Server Fundamentals* has information about how to use persistent caching in applications.

This section contains instructions for administering persistent client caches. It includes the following topics:

- [Enabling and disabling persistent client caching, page 192](#)
- [Creating cache config objects, page 194](#)

Enabling and disabling persistent client caching

Persistent client caching can be controlled at either the repository level or session level.

For a repository

To enable or disable persistent client caching for a repository, set the client_pcaching_disabled attribute in the doabase config object. If persistent client caching is disabled for a repository, users cannot persistently cache objects fetched from the repository or the results of queries against the repository.

The default setting for the attribute is F (FALSE), meaning that persistent caching is enabled for the repository. Setting `client_pcaching_disabled` to T (TRUE) disables persistent client caching for the repository.

For client sessions

There are two `dmcl.ini` keys that control persistent client caching at the session level:

- `client_persistent_caching`
- `cache_queries`

The `client_persistent_caching` key affects both object and query caching. It is TRUE by default, allowing clients to persistently cache both objects and query results. If you set `client_persistent_caching` to FALSE, users cannot persistently cache objects or query results.

The `cache_queries` key affects only the `Cachequery` method and only when `client_persistent_caching` is TRUE. When `client_persistent_caching` is TRUE and `cache_queries` is FALSE, `Cachequery` methods do not cache the query results. The key has no effect on `Query_cmd` methods. The results of any queries issued by a `Query_cmd` method are cached if `client_persistent_caching` is TRUE regardless of the setting of `cache_queries` key. The default value of `cache_queries` is FALSE.

Table 5-4, page 193, shows how these keys and the API methods interact.

Table 5-4. Interaction of persistent caching `dmcl.ini` keys and methods

<code>client_persistent_caching</code> Key Setting	<code>cache_queries</code> Key Setting	Does <code>Cachequery</code> Method Cache Query Results?	Does <code>Query_cmd</code> cache query results (assuming caching argument is set to T)?
T	F	No	Yes
T	T	Yes	Yes
F	F	No	No
F	T	No	No

Changing the `dmcl.ini` keys only affects client sessions started after the changes are saved.

To change the caching behavior for the current API session only, set the `api config` attributes that correspond to the keys. (The attribute names are the same as the keys.) You must set the attributes after initializing the API session and before starting a repository session.

Creating cache config objects

Cache config objects are used to group cached objects and query results into a single unit for purposes of consistency checking. ([Consistency checking, page 49](#), in *Content Server Fundamentals*, describes how cache config objects are used in consistency checking.)

You must be a superuser to create a cache config object.

To create a cache config object:

1. Create a `dm_cache_config` object.
2. Set the `object_name` of the cache config object to a name that is unique among the cache config objects in the repository.
Although not enforced by Content Server, the object name must be unique among the cache config objects in the repository because the object is referenced by name in the API methods. If the name is not unique, consistency checking results will be ambiguous and unpredictable.
3. Set the `cache_element_queries` and `cache_element_types` attributes.
These attributes define the set of cached data managed by the cache config object. Refer to [Defining the cached data set, page 194](#), for information and guidelines for setting these attributes.
4. Set the `server_check_interval`.
This determines how often Content Server validates the cached data. [Defining the server check interval, page 195](#), contains guidelines for setting this attribute.
5. Set the `client_check_interval`.
This determines how often a client application refreshes the cached data. [Defining the client check interval, page 196](#), contains information about how this attribute is used.
6. Save the cache config object.

Defining the cached data set

You must set two attributes to identify the cached data managed by a cache config object: `cache_element_queries` and `cache_element_types`. Both are repeating attributes. The `cache_element_queries` attribute contains a list of DQL queries. Each index position in the attribute can store one DQL SELECT statement. Each SELECT statement can return one or more objects or a set of query results. The `cache_element_types` attribute indicates whether the query in the corresponding index position in `cache_element_queries` is returning objects or a query result set for caching.

The queries can be any legal SELECT statement.

The queries that return objects must identify the same objects that are fetched by the Fetch methods that reference the cache config object. Similarly, the queries that return query results for caching must match the queries defined in Query_cmd methods that reference the cache config object.

If the query is returning objects, include r_object_id and i_vstamp in the selected values list. This ensures that if the object is changed, the results of the query change.

If you are caching objects that are SysObjects or SysObject subtypes, you may want to include the ALL keyword. The ALL keyword directs Content Server to examine all versions of the objects and return any that satisfy the query. If ALL is not included, only the CURRENT version of an object is examined. Including ALL ensures that consistency checking on older object versions is not skipped.

The cache_element_type attribute identifies what the query in the corresponding index position in cache_element_queries returns. Legal values are "query" and "object". For example, suppose you want to cache an SOP named SOP_1216, and to do so, you set cache_element_queries[3] to

```
SELECT "owner_name","i_vstamp","r_object_id" FROM "dm_document"
WHERE "title"='SOP_1216'
```

To tell Content Server that the query in cache_element_queries[3] refers to an object, you must set cache_element_types[3] to object.

The order in which query results are returned is important in consistency checking. If the order of the returned results varies whenever the query is run, it appears as if the results have changed and the client caches are assumed to be invalid. If a query defined in a cache config object is sufficiently complex that the RDBMS may periodically vary the query plan used to execute the query, it is recommended that you include an ORDER BY clause in the query.

Defining the server check interval

The server check interval defines how much time can elapse between validations of the queries in a cache config object. The server check interval expressed as seconds and is defined in the server_check_interval attribute.

When the DMCL issues a CHECK_CACHE_CONFIG administration method, the method sends a request to Content Server to validate the data defined in the cache config object. Content Server subtracts the r_last_checked_date value in the cache config object from the current time and date and compares the result to the value in the server_check_interval. If the subtraction result is greater than the value in server_check_interval, the server calls the dm_CheckCacheConfig method to re-execute the queries defined in the cache config object. The query results are hashed and the

hash is stored in the `i_query_result_hash` attribute and the `r_last_checked_date` and `r_last_changed_date` attributes are updated.

Defining the client check interval

The client check interval determines how often client applications ask Content Server to validate the cached data defined in a cache config object. The client check interval is defined in seconds and set in the `client_check_interval` attribute.

When a session first references a cache config object, the DMCL obtains information about that cache config object from the server and stores that information with a timestamp. Thereafter, whenever the object is referenced during the session, the DMCL subtracts the timestamp from the current time and compares the result to the `client_check_interval` value. If the result is greater than the check interval, the DMCL issues a `CHECK_CACHE_CONFIG` administration method, to ask Content Server whether the cached results are still valid. Content Server uses the value in `server_check_interval` to determine whether or not to rerun the queries in the cache config object.

Manually forcing refreshes

Although data persistently cached in the client DMCL is checked for consistency and automatically updated when needed, you may want to force a refresh sometimes. To force a refresh, you can:

- Flush the objects you want to refresh from a cache
- Reset the `client_pcaching_change` attribute in the `docbase` config object

Flushing a persistent cache

To flush a persistent client cache, use a Flush method. You can flush cached objects or queries or both. Flush only acts on the persistent caches owned by the user who executes the method. You can remove:

- All persistently cached objects and query results in a cache
- Only cached objects or only cached queries from a cache

If a persistent object cache is flushed, the next time the user who owns the cache accesses a persistently cached object, Content Server refetches the object and it is stored in the cache again. Similarly, if cached queries are flushed, the next time the cache owner executes a cached query, Content Server recomputes the results and they are stored again.

in the cache. For instructions on using Flush, refer to [Flush, page 213](#), in the *Content Server API Reference Manual*.

For instructions on using Flush, refer to [Flush, page 213](#), in the *Content Server API Reference Manual*.

Setting the `client_pcaching_change` attribute

The `client_pcaching_change` attribute in the `docbase` config object is used by Documentum clients to determine whether to refresh all persistent client caches. When a Documentum client application starts up, it checks the value of that attribute. If the value has not changed since the last time the value was checked (the last time the application started), the caches are retained. If the value has changed, the DMCL throws away the persistent client caches.

Automating cache config data validation

To automate validation of the cached data defined in a cache config object, create a job to execute the `dm_CheckCacheConfig` method. If you automate the server-side validation, when the DMCL asks the server if the cached data is current, the server can respond immediately, without requiring a check at the time the request is issued.

To create a job automating cache config validations:

1. Create a script that calls the `DO_METHOD` administration method executing the `dm_CheckCacheConfig` method.

The `DO_METHOD` syntax is:

```
apply, session, , DO_METHOD, METHOD, S, dm_
CheckCacheConfig, ARGUMENTS, S, argument_list
```

The argument list must include the following arguments:

- `-docbase_name` *docbase_name*
- `-user_name` *user_name*
- `-cache_config_name` *cache_config_obj_name*

You can also include an argument that sets a tracing level for the method if you like: `-method_trace_level` *trace_level*, where *trace_level* is a value from 0 to 10. If you include the `method_trace_level` argument, you may want to set the `SAVE_RESULTS` argument for the `DO_METHOD` to `TRUE`:

```
apply, session, , DO_METHOD, METHOD, S, dm_
CheckCacheConfig, ARGUMENTS, S, argument_list, SAVE_RESULTS, B, T
```

Doing so causes the trace results to be saved to a document rather than the server log. (For complete information about using `DO_METHOD`, refer to [DO_METHOD](#), page 197, in the *Content Server DQL Reference Manual*.)

2. Create a method object and associate the script with the method object.
Refer to [Chapter 4, Methods and Jobs](#), for instructions on creating a method object.
3. Create a job to execute the method object.
[Chapter 4, Methods and Jobs](#), contains instructions for creating jobs also.
To be most beneficial, schedule the job to execute at an interval more frequent than the interval defined in the `server_check_interval` attribute of the `cache_config_object`.
For general information about creating jobs, refer to [Creating jobs and job sequences](#), page 143.

Overriding consistency checking rules

Overriding the consistency checking rules forces the DMCL to use the default consistency checking rule. The default is to check always. This means that if the cached data is an object, the object is always checked against the object in the repository. If the cached data is a set of query results, the results are always regenerated.

To override consistency checking rules, set the `force_coherency_check` key in the `dmcl.ini` file or the corresponding attribute in the `api config` object or `connection config` object. If you set `force_coherency_check` to `T`, the consistency checking rules defined in `Fetch` and `Query_cmd` methods are ignored by the DMCL. Instead, the DMCL uses the default consistency checking rule.

Where you set `force_coherency_check` to `T` determines the scope of the effect. If you set the key in the `dmcl.ini` file, all sessions started using that `dmcl.ini` file are affected. If you set the attribute in the `api config` object, all repository sessions started withing the context of that API session are affected. If you set the attribute in the `connection config` object, only the repository session represented by the `connection config` object is affected.

Defining the persistent cache write interval

The persistent cache write interval controls how often persistently cached objects are written to the persistent object cache file on the user's local disk during a repository session.

When objects are fetched, they are stored in an in-memory object cache in the client DMCL. The objects are placed in the cache the first time they are fetched by a client. If the object is not persistently cached, it remains in the cache only for the life of the repository

session. If the object is persistently cached, at the close of the session, the object is written to a file on disk. The file is located in:

```
root/object_caches/machine_name/docbase_id/abbreviated_user_name
```

When the next session is started, the objects in the file are loaded back into the in-memory cache.

During a session, persistently cached objects in memory may change. Consequently, each time the in-memory object cache is accessed, the DMCL determines whether the persistent cache write interval has expired and if so, writes all the persistently cached objects from memory to the persistent object cache file. The interval is defined by the `persistent_cache_write_interval` key in the user's `dmcl.ini` file. (The default is 60 minutes.)

The disk file is also updated periodically during sessions, at an interval defined by a `dmcl.ini` key called `client_cache_write_interval`. The value in this key is expressed as seconds and is set to 3600 (60 minutes) by default. The update occurs if any cached object types have been added or changed since the last update. If none of the objects have changed and no new objects have been added to the cache, the update does not occur.

If an explicit transaction is open, writing cached objects to disk is delayed until after the transaction is committed.

Troubleshooting persistent caching

There are several actions that you can take if you need to troubleshoot persistent caching operations:

- Set the API trace level to 10.

Setting the API trace level to 10 lets you see, in the log file, which API calls (Fetch, Query_cmd, and so forth) result in actual server RPC calls. In turn, this tells you which queries are being satisfied using cached data. For example, if a Fetch method has no corresponding RPC call, the cached object was used.

- Examine the cached files directly.

The object cache files and the query result files are stored in ASCII. You can open these files with a text editor.

- Invoke the `dm_CheckCacheConfig` method directly, using IAPI, to test its behavior or results.

Use the `DO_METHOD` administration method to invoke the method. Set the `v-method_trace_level` argument to 10, to generate trace messages, and include the `SAVE_RESULTS` argument on the `DO_METHOD` command line to capture the trace messages in a separate document.

- Execute the `CHECK_CACHE_CONFIG` method with the `FORCE_CHECK` argument set to T (TRUE).

This forces the server to execute the queries defined in the cache config object. You must be a superuser or have Execute Procedure permission on the cache config object to use the FORCE_CHECK argument.

- Flush the caches.

Flushing a cache forces the DMCL to refetch the objects or re-execute the cached queries the next time the objects or query results are accessed. [Flushing a persistent cache, page 196](#), contains information about how to flush a persistent cache.

- Delete or rename the cache files.

After you delete or rename the files, re-execute the Fetch or Query_cmd method to determine if the problem persists. If the problem persists, it is likely that the persistent cache is not source of the problem.

Using persistent client caching with a pre-5.2 Content Server

Persistent client caching is also available for environments using a pre-5.2 Content Server that have installed the 5.2 client DMCL. However, there are some configuration requirements that must be met and a few differences in behavior. The configuration requirements are:

- The dm_cache_config object type must be created in the repository.
- The dm_CheckCacheConfig method must be installed in the repository.

This means that the dm_method object representing the dm_CheckCacheConfig method must be created in the repository and the method's executable (dm_CheckCacheConfig.exe) must be installed in the Content Server's bin directory.

If you are installing the 5.2 Desktop Client DocApp in the repository, the DocApp installation procedure creates the dm_cache_config object type and creates the method object. You can copy the method's executable from a 5.2 Content Server installation. The method is also packaged with the 5.2 Desktop Client DocApp available from the product download site.

In a 5.2 repository, the docbase config object has two attributes not available in pre-5.2 repositories: client_pcaching_disabled and client_pcaching_change attributes. The client_pcaching_disabled attribute turns persistent client caching on and off. The client_pcaching_change attribute is used to force a full refresh of all client caches.

If you are running Desktop Client against a pre-5.2 repository, control persistent client caching in the following manner:

- Use the client_caching_enabled key in the dmcl.ini to turn caching on and off

This turns persistent caching on or off for all sessions using the dmcl.ini. It is not repository-specific.

- Use the `effective_date` attribute in the `docbase config` attribute to force a refresh of the caches

Changing the value in this attribute will force the DMCL to flush and refresh all caches for clients in that repository.

Note: Setting or changing the `effective_date` attribute has no effect in a 5.2 repository.

Configuring DMCL exception handling



Caution: This section contains general information about enabling DMCL exception handling. If the host machine is a UNIX platform, refer to the release notes for specific information about this feature on your platform.

This feature is not available on Linux platforms.

By default, when a DMCL exception occurs, the client application using the DMCL is terminated. (An exception is an error that causes a program or application to terminate. For example, an access violation on Windows or a segmentation violation on UNIX is an exception.) There are two ways to configure the DMCL to change this behavior:

- You can use the `terminate_on_exception` key in the `dmcl.ini` file to stop the exception from terminating the client application.
- You can configure the DMCL to continue on exception.

Note: If heap corruption is detected when handling an exception, the DMCL is always shut down even if continue on exception behavior is enabled.

Configuring the default behavior

The `terminate_on_exception` key in the `dmcl.ini` file controls whether the DMCL shuts down and the client application using it is terminated when an exception occurs. By default, this key is T (TRUE).

If you set the key to F (FALSE), when an exception occurs, the DMCL is shut down, and all DMCL calls except `Getmessage` will fail. However, the application using the DMCL is not terminated.

Enabling the DMCL to continue on exception

Use the `exception_count` and `exception_count_interval` keys in the `dmcl.ini` file to enable the DMCL to continue on exception. When set, these keys interact to determine how

many exceptions can occur within a specified interval before the DMCL is automatically shut down.

For example, suppose you set `exception_count` to 5 and `exception_count_interval` to 30 minutes. The DMCL will keep a record of the 6 most recent exceptions. When an exception occurs, the DMCL checks the how many exceptions are recorded. If the number is the value of `exception_count` plus one, in this case 6, the oldest exception is discarded and the new exception is added to the record. If the number is less than `exception_count` plus one, the new exception is simply added to the list.

Then, the DMCL compares the time and date stamp on the oldest and the newest exceptions in the list. If the time difference is equal to or greater than the value set in `exception_count_interval` (30 minutes in our example), the DMCL is allowed to continue. If the time difference is less than the interval specified in `exception_count_interval`, the default behavior is invoked.

Depending on the setting of the `terminate_on_exception` key, the default behavior is either:

- Allow the exception to terminate the client application
- Allow the client application to continue but shut down the DMCL so that all calls fail with an error.

Note: On Windows, when handling a DMCL exception, if heap corruption is detected, the DMCL is always shut down regardless of the settings for continue on exception.

Set the `exception_count` and `exception_count_interval` keys before opening any session using the `dmcl.ini`.

Setting these keys after the DMCL is shutdown has no effect on the shutdown. It will not re-enable the DMCL.

Generating DMCL exception trace files

When a DMCL exception occurs, a description of the exception and a DMCL stack trace is written to a file. The file is named:

```
dmcl_err_pidpid_number_date_time.txt
```

Where the file is stored on the filesystem is determined by the `trace_file` key in the `dmcl.ini` file.

- If `trace_file` is not set, the file is stored in the current working directory of the client process.
- If `trace_file` is set to a directory path (with or without inclusion of a file name), the file is stored in the last directory represented by the path.

For example, if `trace_file` is set to `/mydirectory/traces` or to `/mydirectory/traces/trace.log`, the file is stored in `/mydirectory/traces`.

- If `trace_file` is to only a file name, the DMCL stack trace file is stored in the current working directory of the process.

Connection Brokers

This chapter contains information about the connection broker, Documentum's name server. The chapter includes the following topics:

- [An overview of connection brokers, page 205](#)
- [Servers and connection brokers, page 208](#)
- [Clients and connection brokers, page 208](#)
- [Failover for connection brokers, page 209](#)
- [Load balancing for connection brokers, page 209](#)
- [Configuring a connection broker, page 209](#)
- [Restarting a connection broker, page 213](#)
- [Starting additional connection brokers, page 214](#)
- [Shutting down a connection broker, page 215](#)
- [Obtaining information from a connection broker, page 218](#)
- [Obtaining Information about Connection Brokers, page 219](#)
- [Deleting server information, page 220](#)

An overview of connection brokers

The Documentum connection broker is a process that provides client sessions with connection information. To establish a connection, a client session must know where to find a server that accesses the requested repository. When a client session is opened, the client contacts the connection broker and requests the information it needs to connect with a server for the requested repository. The connection brokers that can handle a client's connection request are defined in the client's dmcl.ini file. ([Defining connection brokers for connection requests, page 164](#), contains more information about defining connection broker targets for clients.)

The connection broker sends back the IP address for the host on which such a server resides and the port number that the server is using. If there are multiple servers for the

repository, the connection broker returns connection information for all of them. The client session then uses that information to choose a server and open the connection.

Clients can request a connection through a particular server, any server on a particular host, or a particular server on a specified host. ([Requesting a specific server connection, page 167](#), contains instructions.)

How many connection brokers are there?

By default, each installation must have one connection broker. The first connection broker is started as part of the installation process. You can set up additional connection brokers later if you like. ([Starting additional connection brokers, page 214](#), contains instructions.)

What information does a connection broker have?

Each connection broker has information about servers and the repositories they access. Server information includes:

- The server's name
- The process ID
- The name of the host machine on which the server resides
- The server's status
- When the connection broker last received some status information
- When the connection broker expects to hear from the server again

Repository information includes:

- The repository name and ID
- A verbose description

System administrators can review connection broker information by issuing a `Getdocbasemap` or `Getservermap` method call. `Getdocbasemap` returns the information that a connection broker has for each repository known to it. `Getservermap` returns the information that a connection broker has for each server known to it. ([The `Getdocbasemap` method, page 218](#), and [The `Getservermap` method, page 218](#), respectively, contain instructions for using these methods.)

How does a connection broker get information?

Each server broadcasts information to connection brokers at regular intervals. The broadcast contains the information maintained by connection brokers about the server and the repository accessed by the server. ([Servers and connection brokers, page 208](#), contains a detailed description of the communications between servers and connection brokers.)

How do I locate connection brokers?

You can find out which connection brokers a client can talk to by issuing a `Getdocbrokermap` method call. This method returns a `docbroker locator` object. A `docbroker locator` object is a non-persistent object that is constructed and returned by the client library (`dmcl`) in use by the session. It tells you which connection brokers your session can access. For each connection broker, it tells you the host on which the connection broker resides, what communication protocol it uses, the port number it uses for communication, and its time-out period. ([Obtaining Information about Connection Brokers, page 219](#), contains the procedure.)

Alternatively, you can also examine the client's `dmcl.ini` file.

Connection broker configuration options

A standard connection broker needs no additional configuration. It works as installed. However, you may want to change the configuration to:

- Protect the connection broker from being shutdown by an unauthorized person
- Define which servers can access the connection broker
- Define a specific IP listening address for the connection broker

This is one way you can run multiple connection brokers on one machine.

- Enable a connection broker to service connection requests that originate outside a firewall

All of these options are set up in the connection broker initialization file. This is an optional file that is referenced on the command line when you start a connection broker. [Configuring a connection broker, page 209](#), contains instructions on the format of the initialization file, and how to configure the options.

Servers and connection brokers

Connection brokers do not ask servers for information. Instead, they wait for servers to broadcast information to them.

When you start a server, it automatically broadcasts information about itself. Each connection broker that receives the broadcast adds the server to its registry, or list of known servers. The server sends the first broadcast before it is fully initialized, so the connection broker sets the server's status to starting. As soon as the server is fully initialized and ready to service clients, it broadcasts a checkpoint message. The receiving connection brokers then update the server's status to open.

Thereafter, the server broadcasts a checkpoint message at regular intervals. By default, this checkpoint interval is five minutes. However, you can reset it to a different value ([Setting the checkpoint interval, page 114](#), contains instructions).

A connection broker keeps track of when it last heard from a server and when it expects to hear from that server next. If it does not receive a broadcast from a server at the expected time, it sets the server's status to presumed down. A connection broker keeps the entry for a non-broadcasting server for a certain number of hours. At the end of that time (the keep entry interval), if the connection broker has not heard from the server, it removes the server from its registry. By default, the keep entry interval is 24 hours. However, you can change it ([Setting the keep entry interval, page 116](#), contains instructions).

When a system administrator shuts down a server, the server sends out a message telling the connection brokers that it is going down. The connection brokers that hear that message set the server's status to stopped. Later, when the server is restarted, it rebroadcasts its information and the connection brokers update their entries for the server and reset its status.

Clients and connection brokers

A client is anything requesting a connection to a server. A client can be Desktop Client™, an external application, or a user working with IDQL or IAPI, for example. In each case, at the start of a client session, a Connect method must be executed to establish a connection to a repository. The Connect method's first operation is to send a message to a connection broker asking for the service information that it needs to make the requested connection.

Each client session has a default connection broker. Generally, to ensure continuous service, clients have backup connection brokers also.

The default and backup connection brokers for clients are defined in the dmcl.ini file. For ease of administration, this file is often put in one central location and referenced by

each client in a local file. ([Defining connection brokers for connection requests](#), page 164, contains information about setting up this file.)

Failover for connection brokers

Failover for connection information requests from a client is provided by identifying one or more backup connection brokers in the dmcl.ini file and ensuring that the auto_request_forward key in the dmcl.ini file is set to T. You must explicitly define the back up connection brokers in the dmcl.ini file. The auto_request_forward key is T by default. (For instructions on defining backup connection brokers, refer to [Format of dmcl.ini sections](#), page 165.)

By default, a connection request is handled by the primary connection broker. However, if that connection broker does not respond within 15 seconds or less or does not know about the requested repository, failover occurs if a backup connection broker is defined and auto_request_forward is T. The request is forwarded to the first backup connection broker. If that connection broker does not respond, the request is forwarded to the second backup connection broker. The request goes in turn to each backup connection broker known to the client until a connection broker responds successfully or until all connection brokers defined in the dmcl.ini file have been tried. If there is no successful response, the request fails.

Load balancing for connection brokers

To ensure that connection requests do not all fall on the connection broker identified as the primary connection broker in a dmcl.ini file, use the docbroker_search_order key in the dmcl.ini file. The docbroker_search_order key determines whether the request is sent first to the primary connection broker or to a connection broker randomly selected from the list of connection brokers in the dmcl.ini file.

Setting this key to “random” causes the DMCL to randomly select a connection broker from those specified in the dmcl.ini file for a connection request. Randomly selecting a connection broker from the list of connection brokers in a dmcl.ini file helps prevent connection broker performance bottlenecks that can occur if large numbers of users are sharing the same connection brokers.

Configuring a connection broker

You can configure a connection broker to include the following options:

- Shutdown security (UNIX only)
You can configure a connection broker to require a password whenever the connection broker is shut down.
- Server access restrictions
You can configure a connection broker to accept or reject broadcasts from specific servers.
- Specific IP listening addresses
You can define specific IP listening addresses for connection brokers, which allows you to run multiple connection brokers on one machine by using a specific network card for each connection broker. [Starting additional connection brokers, page 214](#), contains instructions on this option.
- IP address translation
You can configure a connection broker to receive connection requests from clients outside a firewall and direct the request to the appropriate repository.

All of these features are configured in an initialization file, which is then referenced on the connection broker's start-up command line.

Connection broker initialization file

The connection broker initialization file is an optional file that you create. It has the following format:

```
[SECURITY]
password = string_value
allow_hosts=host_name_list|deny_hosts=host_name_list

[DOCBROKER_CONFIGURATION]
host = host_name|IP_address_string
service = service_name
port = port_number

[TRANSLATION]
port=["]inside_firewall_port=outside_firewall_port
{,inside_firewall_port=outside_firewall_port"}["]
host=["] inside_firewall_port=outside_firewall_port
{,inside_firewall_port=outside_firewall_port"}["]
```

Note: The [SECURITY] section is recognized and used only on UNIX platforms. Connection brokers on Windows platforms are not affected by the definition of a security password.

If a valid service name is included, the connection broker is started with the service name. Otherwise, if a valid port number is included, the connection broker is started using the port number. If neither is included, the connection broker is started on port 1489.

The file can have any valid file name. You can store the file in any location, but the most convenient place may be the same directory in which the `dm_launch_docbroker` script is stored.

The translation strings are enclosed in double quotes if multiple ports or hosts are specified.

Invoking the initialization file

When you start a connection broker, the initialization file is not automatically invoked. To invoke the file, you must include the `-init_file` argument on the start-up command line. For Windows platforms, the syntax is:

```
drive:\documentum\product\version_number\bin\dmdocbroker.exe
-init_file filename
```

If the connection broker is running as a service, edit the service entry to include the argument on the command line. You can use the Documentum Server Manager to edit the service entry.

For UNIX platforms, the syntax is:

```
% dm_launch_docbroker -init_file filename
```

If there are other arguments on the command line in addition the initialization file argument, the `-init_file` argument must appear first or it is ignored.

Configuring shutdown security (UNIX only)

Defining a security password for a connection broker ensures that only a user who knows the password can stop the connection broker. Define a password in the [SECURITY] section with the password key:

```
[SECURITY]
password=string_value
```

Restricting server access

By default, a connection broker accepts broadcasts from any server. However, you can configure a connection broker to either:

- Accept broadcasts only from specified servers
- Reject broadcasts from specified servers

To define accepted servers, use the following format in the initialization file:

```
[SECURITY]
...
allow_hosts=host_name_list
```

To define rejected servers, use the following format:

```
[SECURITY]
...
deny_hosts=host_name_list
```

host_name_list is a list of the host machines on which the servers reside. Multiple host names are separated by commas. For example:

```
[SECURITY]
...
deny_hosts=bigdog,fatcat,mouse
```

The options are mutually exclusive. For each connection broker, you can configure either the accepted servers or the rejected servers, but not both.

Translating IP addresses

For security reasons, many enterprises set up firewalls to prevent outside users from accessing enterprise repositories and file systems. However, on occasion, users outside the firewall may need to access a repository inside the firewall.

To allow a user or client application to connect to a repository inside a firewall, the connection broker initialization file includes a [TRANSLATION] section. This section redirects a request to a safe IP address and port name. The section has the following format:

```
[TRANSLATION]
port=outside_firewall_port=inside_firewall_port
{,outside_firewall_port=inside_firewall_port}
host=outside_firewall_IP=inside_firewall_IP
{,outside_firewall_IP=inside_firewall_IP}
```

outside_firewall_port and *inside_firewall_port* are port numbers.

outside_firewall_IP and *inside_firewall_IP* are IP addresses.

For example, suppose repository A is inside a firewall and that application B, outside the firewall, wants to connect to repository A. Also suppose the connection broker that receives the request has the following TRANSLATION section in its initialization file:

```
[TRANSLATION]
port=2231=4532
host=2.18.13.211=172.18.23.257
```

When the connection broker receives the request, it translates 2231 and 2.18.13.211 to 4532 and 172.18.23.257. It sends the values 4532 and 172.18.23.257 back to application B, which uses them to establish the connection.

If you specify multiple ports or hosts for translation, you must enclose the translation string in double quotes. For example:

```
[TRANSLATION]
port="2253=6452,22254=6754"
```

To use a [TRANSLATION] section:

1. Define the IP address translation rules in the firewall.
2. Reflect these rules in the [TRANSLATION] section of the connection broker's initialization file.
3. Restart the connection broker, specifying the initialization file on the command line.

Restarting a connection broker

The procedures in this section describe how to restart a connection broker that was stopped.

Windows platforms

A connection broker running on a Windows platform may be running as a service or may have been started from the command line.

To start a connection broker running as a service:

1. Double-click Start connection broker in the Documentum group.
This launches the connection broker executable.

To start a connection broker that is not running as a service:

1. At the operating system prompt, enter the startup command line.

The syntax for the startup command is:

```
dmdocbroker.exe [-init_file filename] -host host_name
-service service_name -port port_number
```

For example:

```
d:\documentum\product\4.0\bin\dmdocbroker.exe
-init_file DocBrok.ini -host engr -service engr_01
```

UNIX platforms

Use the following procedure to restart a connection broker that runs on a UNIX platform.

To restart a connection broker:

1. Log into the machine on which you want to start the connection broker.
2. Change to the \$DOCUMENTUM/dba directory:

```
% cd $DOCUMENTUM/dba
```
3. At the operating system prompt, type the command line for the `dm_launch_docbroker` utility.

The command line syntax is:

```
dm_launch_docbroker [-init_file file_name] [-host host_name]  
[-service service_name] [-port port_number]
```

You must include the host name and a service name or port number to identify the connection broker unless you specify an initialization file that includes a [DOCBROKER_CONFIGURATION] section to identify the connection broker.

Starting additional connection brokers

You can run multiple connection brokers for a repository. The connection brokers can run on separate machines. You can also run multiple connection brokers on the same machine. If you choose to run multiple connection brokers on the same machine, each connection broker on the machine must use a separate port or a separate network card.

To configure a connection broker to use a separate port, define a services file entry that identifies the service name for the connection broker. The service name for the connection broker must be unique among the service names. Alternatively, create an initialization file that identifies the service. For example:

```
[DOCBROKER_CONFIGURATION]  
host=host_machine_name  
service=service_name
```

or

```
[DOCBROKER_CONFIGURATION]  
host=host_machine_name  
port=port_number
```

where *port_number* is the port identified in the new service.

To configure a connection broker to use a separate network card, create an initialization file for the connection broker. The file must include a

[DOCBROKER_CONFIGURATION] section to identify the IP address of the network card. Use the following format:

```
[DOCBROKER_CONFIGURATION]
host=IP_address_string
service=service_name
port=port_number
```

IP_address_string must be in the dotted decimal format (for example, 143.23.125.65).

The service name is the connection broker's service name, defined in the host machine's services file. The port number is the port defined in the service. (Refer to *Content Server Installation Guide* for instructions on setting up service names.) If you include a service name, the connection broker is started using that service. Otherwise, if you include a port number, the connection broker is started using that port. If you do not include a service name or a port number, the connection broker uses port number 1489.

To start additional connection brokers:

1. Start the connection broker from the operating system prompt.

- On Windows:

```
d:\documentum\product\4.0\bin\dmdocbroker.exe
[-init_file filename ]-host host_name -service service_name
```

- On UNIX:

```
dm_launch_docbroker [-init_file filename]-host host_name
-service service_name
```

You must include the host name and a service name or port number to identify the connection broker unless you specify an initialization file that includes a [DOCBROKER_CONFIGURATION] section to identify the connection broker.

2. In the server config object, modify the projection attributes to add the new connection broker as a server projection target.
3. Optionally, modify the server.ini file to add the new connection broker as a server projection target.

Refer to [Defining connection broker projection targets, page 112](#), for information about adding the new target to the server.ini file.

4. Reinitialize the server.

Shutting down a connection broker

The procedures in this section describe how to shut down a connection broker. If the connection broker was configured with shutdown security, you must know the correct password to shutdown the connection broker.

Windows platforms

A connection broker on a Windows host may have been started as service or may have been started from the command line.

To stop a connection broker that is running as a service:

1. Click **Start**→**Program Files**→**Documentum**→**Server Manager**
2. Select the connection broker tab.
3. Select the correct connection broker.
4. Click **Stop**.

To stop a connection broker started from the command line:

1. Close the window in which the connection broker is running.

UNIX platforms

A connection broker started on a UNIX host is started from the command line.

To stop a connection broker started from the command line:

1. Execute the `dm_stop_docbroker` utility. The command line for this utility is:

```
% dm_stop_docbroker [-Ppassword] [-B[atch]]  
[-Nport_number] [-Sservice_name]
```

The utility stops the connection broker that is running on the host specified in the `dmshutdown` script. The connection broker specified in that script is set when you run `dm_setup` during the initial server installation. (It will be the connection broker with which your first-installed server is communicating.) If you execute `dm_stop_docbroker` in the interactive mode (without the `-B` flag), the utility tells you which connection broker it intends to stop and prompts you for confirmation. If you include the `-B` flag, the utility does not prompt for confirmation or tell you which connection broker it is stopping. The default for the `-B` flag is `FALSE`.

You can include the `-N` and `-S` arguments to identify a particular connection broker to shut down if you have multiple connection brokers on one machine.

The *password* is the password specified in the connection broker initialization file. You must supply this password if the connection broker was started with security imposed. ([Configuring shutdown security \(UNIX only\)](#), page 211, contains information about imposing security on connection brokers.)

If, for some reason, you cannot use the `dm_stop_docbroker` utility, you can use the Unix `kill` command to stop the connection broker if it was started without security. (If you do not know the process ID for the connection broker, you can obtain it using

the Unix ps command.) You cannot use the Unix kill command to stop a connection broker that was started with security. Only the Unix kill -9 command will stop a secured connection broker.

Multiple connection brokers on UNIX

If you have multiple connection brokers, you can stop two or more by editing the `dm_stop_docbroker` script before running the `dm_stop_docbroker` utility. The script is found in `$DOCUMENTUM/dba`. The final line in this script identifies the host of the connection broker that will be stopped when the `dm_stop_docbroker` utility is run. Here is an example of the code line:

```
./dmshutdown docbroker -Tlapdog -P$password $@  
# lapdog is the host name
```

To stop multiple connection brokers, modify the script by adding multiple copies of this line, one for each host on which a connection broker resides. For example, suppose there are connection brokers running on `lapdog`, `fatcat`, and `mouse`. If you wanted to stop all three with one iteration of `dm_stop_docbroker`, edit `dm_stop_docbroker` to include these three lines:

```
./dmshutdown docbroker -Tlapdog -P$password $@  
#lapdog is the host name  
  
./dmshutdown docbroker -Tfatcat -P$password $@  
#fatcat is the host name  
  
./dmshutdown docbroker -Tmouse -P$password $@  
#mouse is the host name
```

The `dm_stop_docbroker` utility substitutes the password that you specify on its command line for `$password` in the script. If you imposed a different password for each connection broker, you must explicitly put the correct password in the command line that shuts down each connection broker:

```
./dmshutdown docbroker -Tlapdog -P$password $@  
#lapdog is the host name  
  
./dmshutdown docbroker -Tfatcat -Pmeowmeow $@  
#fatcat is the host name  
  
./dmshutdown docbroker -Tmouse -Psqueak $@  
#mouse is the host name
```

Obtaining information from a connection broker

A connection broker can provide information about the servers known to it or about the repositories the servers access. To obtain this information, a client (such as an application or a system administrator) must issue the `Getservermap` or `Getdocbasemap` method. The `Getservermap` method returns server information for a particular repository, and `Getdocbasemap` returns information about repositories.

The `Getservermap` method

Use the `Getservermap` method to obtain information about the servers known to a connection broker that access a particular repository. (Refer to [Getservermap](#), page 256, in the *Content Server API Reference Manual* for the syntax.)

By default, the method is sent to the primary connection broker for your session. However, if you want to ask a particular connection broker for information, the method arguments let you define an alternate recipient connection broker.

The `Getservermap` method returns an object ID for a server locator object. A server locator object is a non-persistent object that contains the information known to the connection broker about any server for the repository. ([Server Locator](#), page 437, in the *EMC Documentum Object Reference Manual* contains a description of the attributes in a server locator object.)

Use the `Get` method with the server locator's ID to retrieve the values in the attributes. The information for a single server appears at corresponding index positions in the repeating attributes. For example, the name of the host machine for the server named in `r_server_name[0]` is found in `r_host_name[0]`, its process ID is found in `r_process_id[0]`, and its status is found in `r_last_status[0]`.

The `Getdocbasemap` method

Use the `Getdocbasemap` method to obtain information about the repositories that servers can access. (Refer to [Getdocbasemap](#), page 235, in the *Content Server API Reference Manual* for the syntax.)

By default, the method is sent to the primary connection broker for your session. However, if you want to ask a particular connection broker for information, the method arguments let you define an alternate recipient connection broker.

The `Getdocbasemap` method returns an object ID for a docbase locator object. A docbase locator object is a non-persistent object that contains information about the repositories associated with the servers known to the connection broker. ([Docbase Locator, page 191](#), in the *EMC Documentum Object Reference Manual* contains a description of the attributes in a docbase locator object.)

Use the `Get` method with the docbase locator ID to retrieve the values in the attributes. The information for a single repository appears at corresponding index positions in the repeating attributes. For example, the name of the repository whose ID is in `r_docbase_id[3]` is found in `r_docbase_name[3]` and its description is found in `r_docbase_description[3]`.

Obtaining Information about Connection Brokers

Sometimes, you may want to know which connection brokers a client can access. There are two ways that you can get that information:

- Use the `Getdocbrokermmap` method
- Use the `Get` method to obtain the information from the api config object

Using the `Getdocbrokermmap` method

The `Getdocbrokermmap` method returns a docbroker locator object that lists, in repeating attributes, all connection brokers that are visible to the client session. ([Getdocbrokermmap, page 237](#), in the *Content Server API Reference Manual* contains the syntax of `Getdocbrokermmap`.)

The information in the docbroker locator object's attributes is taken from the `dmcl` file that the client session is accessing. The information for a single connection broker appears at corresponding index positions in the repeating attributes. For example, the values at `network_protocol[2]`, `host_name[2]`, `port_number[2]`, and `time_out[2]` describe one connection broker.

`Getdocbrokermmap` is also useful when you want to send a `Getdocbasemap` or `Getservermap` method to a particular connection broker and need to find the protocol, host name, and port number values for the connection broker.

Using the Get method

For each client session, there is a non-persistent api config object that contains configuration information about the session. The information in this object is taken from the dmcl.ini file in use by the session and identifies the primary and back up connection brokers for the session. To get this information, use the following syntax:

```
dmAPIGet("get, session, apiconfig, attribute_name[[index]]")
```

The apiconfig keyword is an alias for the api config object. Because you can specify only one attribute with each execution, you must execute the Get method multiple times to obtain all of the information.

The information about the primary connection broker is contained in four single-valued attributes (primary_protocol, primary_host, primary_port, and primary_timeout). The information for the backup connection brokers, if any, is contained in four repeating attributes (backup_protocol, backup_host, backup_port, and backup_timeout). [Api Config, page 68](#), in the *EMC Documentum Object Reference Manual*, contains a description of these attributes.

Deleting server information

A connection broker deletes a server from its list of known servers when either of the following situations occurs:

- A server sends a delete me message as a result of a manual shutdown.
This situation occurs when a server is shutdown with instructions to broadcast a delete me message. ([Using the Shutdown method, page 119](#), contains instructions.)
- A server fails to broadcast a checkpoint message within the expected keep_entry interval.
This situation occurs when a server is not active as a result of a shutdown without a delete me message or a crash or when the network between the server and connection broker fails.
- A server is reinitialized after a change to the projection targets in the server config object that deletes the connection broker from the targets.

Content Management

Content is the text, graphics, tables, charts, and other information, excepting attribute values, that is stored with an object. Content is typically stored in documents or a document subtype. However, any SysObject or SysObject subtype except cabinets and folders accepts content. Content associated with an object is stored in a storage area defined in the repository. This chapter describes how to create, use, and manage those storage areas. The chapter covers the following topics:

- [Storage area options, page 221](#)
- [Summary of storage area configuration options, page 234](#)
- [Content and full-text indexes, page 235](#)
- [How objects, contents, and storage are connected, page 236](#)
- [Content retention, page 243](#)
- [System-defined storage areas, page 244](#)
- [File paths and URLs for content files in storage, page 244](#)
- [Setting up storage, page 246](#)
- [Providing automatic file extensions, page 262](#)
- [Moving content files, page 262](#)
- [Maintenance operations for storage areas, page 266](#)
- [Administering content assignment policies, page 279](#)
- [Archiving and restoring documents, page 280](#)

Note: This chapter does not describe distributed storage areas in detail, nor how to set up or manage distributed storage areas. They are described in detail, including their setup and management, in the *EMC Documentum Distributed Configuration Guide*.

Storage area options

There are a variety of storage area options supported by Content Server. Which storage option or options you choose for a repository depends on what kinds of files you plan to

store in the repository, how you intend to use those files, and how you have designed the storage strategy in your installation. The options are:

- File system directories

Storing content in file system directories is the most common choice. The file store, distributed store, and linked store storage options all use file system directories as the underlying storage facility. [File store storage areas, page 223](#), describes file store storage areas in detail.

- Content-addressed storage

Content in a content-addressed storage is stored in a separate (third party) content-addressed storage system. To use content-addressed storage, you must have purchased a Content Services for EMC Centera license. [Content-addressed storage, page 227](#), describes content-addressed storage areas.

Note: The Content Services for EMC Centera license is not supported on the HP Itanium platform.

- Blob storage

Content in blob storage is stored directly in the repository, in a special table. [Blob store storage areas, page 230](#), describes blob storage areas.

- Turbo storage

Content in turbo storage is stored in an attribute in content or subcontent objects. [Turbo storage, page 231](#), describes turbo storage areas.

- Distributed storage

Content in distributed storage areas is replicated from its primary location to all component storage areas of the distributed storage area. Using distributed storage areas is beneficial if the business has sites in different geographic locations.

The components of a distributed storage area can be file store storage areas, content-addressed storage areas, or linked store storage areas.

Setting up and managing a distributed storage area is described in the *EMC Documentum Distributed Configuration Guide*.

- External storage

Content in external storage is stored on external storage devices outside the Documentum system. [External storage areas, page 232](#), describes this option.

- Linked storage

A linked storage area points to an underlying storage area, typically a file store storage area. [Linked store storage areas, page 233](#), describes the use of a linked store storage area.

With the exception of turbo storage, all the storage options are represented in the repository by specific object types that are subtypes of the dm_store object type. The store

object type provides a set of attributes inherited by all the storage subtypes. [Table 2–159, page 453](#), in the *EMC Documentum Object Reference Manual* lists these common attributes.

File store storage areas

File store storage areas are the basic building blocks of a storage strategy. Storage areas of this type can contain content files in all formats. In most installations, the majority of the content files are stored in file store storage areas. A file store storage area is represented in the repository as an object of type `dm_filestore`. [File Store, page 245](#), in the *EMC Documentum Object Reference Manual* lists the attributes defined for the file store object type.

File store storage areas offer a variety of configuration options, including:

- Private or public access
- Content encryption
- Content compression
- Duplicate-content checking and prevention
- Digital shredding

Note: The compression and duplicate-content checking and prevention options require a Content Storage Services license. The digital shredding option requires a Trusted Content Services license.

Public and private file store areas

You can define a file store storage area as public or private.

If a storage area is defined as public, the server assumes that all files stored in that area are open to the public. This means that the files, as well as the directories that contain them, must be public at the operating system level. When a client fetches a file from a public storage area, the file is not copied to a mutually accessible area. Instead, the server gives the client access to the file directly, in the storage area.

If a storage area is defined as private, the server does not provide direct access to the file in the storage area. Instead, when a client fetches a file from a private storage area, the server copies the file to some user-defined location that is mutually accessible to both the server and the client.

File store storage areas are private by default. Use Documentum Administrator to change the file store storage area to private. The public or private setting is recorded in the `is_public` attribute of the storage area's `dm_filestore` object. The attribute is set to T

(TRUE) if the storage area is a public storage area and to F (FALSE) if the storage area is private.

Content encryption

Content encryption is a configuration option only if you have installed Content Server with a Trusted Content Services license. The option allows you to designate a file store storage area as an encrypted storage area when you create the storage area. The setting for encryption cannot be changed after the storage area is created. You cannot change an encrypted storage area to a non-encrypted storage area, nor can you change a non-encrypted storage area to an encrypted storage area.

If a storage area is configured to use content encryption, all content files stored in that storage area are encrypted. Content Server automatically encrypts the content when it is saved to the storage area and decrypts the content when it is retrieved from the area. For a complete overview of the implementation of encrypted storage areas, refer to [Encrypted file store storage areas, page 103](#), in *Content Server Fundamentals*.

It is possible to both encrypt and compress content in a file store storage area. In such cases, Content Server compresses the content first and then encrypts it.

Content compression

This option requires Content Storage Services license.

The content compression option directs Content Server to compress content saved to the storage area. If this option is enabled, Content Server automatically compresses content written to the storage area and decompresses the content when retrieving the content.

Content compression is enabled when the storage area is created and cannot be reset later.

Note: If your clients are configured to use compression (`use_compression` in the `dmcl.ini` file is set to T), Content Server uncompresses the content and applies the compression algorithm used for the storage area if saving content to a storage area configured for compression.

Content duplication checking and prevention

The content duplication checking and prevention option is used to prevent Content Server from saving duplicate content files to a storage area. This option is particularly useful if you plan to store archived email or similar content in the storage area. For

example, suppose you are storing email archives in a storage area. If your company has 10,000 employees and each receives the same email attachment, there is no need to store 10,000 copies of the attachment in the storage area. If the content duplication checking and prevention option is set for the storage area, Content Server saves one copy of the attachment to the storage area and then creates content objects for the other 9,999 recipients that point to the originally saved copy.

Note: Content duplication checking and prevention is not applied, even if configured for a storage area, to Macintosh resource fork content files stored in the storage area.

How checking and prevention work

If a storage area has content duplication checking and prevention configured, Content Server generates hash values for content files stored in the storage area. The hash value for each content file is stored in an attribute of the file's associated content object. Before saving a content file to the storage area, the server compares the file's generated hash value to the hash values recorded for content of the same format already in the storage area. If it finds a duplicate hash value for a content file of the same format, the server creates a new content object for the content file but does not actually write the content to the storage area. Instead, it points the new content object at the existing content file.

It is possible to configure a storage area so that Content Server generates the hash files but does not perform the comparison operation. If you configure a storage area in that manner, Content Server writes the content to the storage area even if a copy of a content already exists in the storage area.

Supporting attributes

Three attributes support this functionality:

- `content_hash_mode`
- `content_dupl_pref`
- `r_content_hash`

`content_hash_mode` and `content_dupl_pref`

The content duplication checking and prevention setting for a storage area is recorded in two attributes of the storage area: `content_hash_mode` and `content_dupl_pref`. These attributes are defined for `dm_store` and inherited by `dm_filestore`.

The `content_hash_mode` attribute, if set to 1, causes Content Server to generate a hash value for a content file saved to the storage area. The hash value is stored in the content

where `digestedData` is the SHA1 hash string returned by the crypto library.

Tracing duplication checking and prevention

If the `DM_SERVER` tracing facility is turned on, messages are recorded in the log file whenever a content file is found to be a duplicate of an existing content in a storage area. (The `DM_SERVER` tracing facility is turned on using the `Trace` method.)

Digital shredding

Digital shredding is a process that ensures that a content file is deleted in an unrecoverable way. You must have installed Content Server with a Trusted Content Services license to enable digital shredding.

If digital shredding is enabled for a file store storage area, when a document whose content is stored in that storage area is deleted from the repository, Content Server immediately digitally shreds the content and removes the document and the associated content object from the repository.

Content Server utilizes the capabilities of the underlying operating system to shred content files. The shredding algorithm is in compliance with DOD 5220.22-M (NISPOM, National Security Industrial Security Program Operating Manual), option d. This algorithm overwrites all addressable locations with a character, then its complement, and then a random character.

You can enable digital shredding for any standalone file store area. You may also enable shredding for file store storage areas that are the targets of linked store storage areas. The feature is not supported for file store storage areas that are components of a distributed storage area.

Content-addressed storage

Note: Using content-addressed storage requires a Content Services for EMC Centera license. This license is available on all supported platforms except HP Itanium. If you install Content Server with that license, Content Server supports the use of the EMC Centera Content-Addressed Storage System. (Refer to the release notes for the exact version.)

Use content-addressed storage systems if you want to store massive amounts of unchanging data, such as email archives or check images. In addition to the content, a content-addressed storage system allows you to store up to 62 metadata values with each piece of content in the system.

Note: You cannot store files created on a Macintosh machine in a content-addressed storage area.

Content in a content-addressed storage area is located using a content address rather than a file path. The address is created and managed by the storage system and provided to Content Server. A particular content file may have multiple addresses. Some operations generate new addresses for the content. For example if you change the metadata values stored with the content, the system generates a new address for the content. The content address is stored in the content object. Additional addresses are stored in the `i_contents` attribute of an associated `dmi_subcontent` object.

Content Server communicates with the storage system through a plug-in shared library provided with Content Server. The library invoked by the plug-in is also installed with Content Server.

In the repository, a content-addressed storage area is defined using a `ca store` object. The storage object identifies the content metadata and values that you want to store with content in the storage system. It also defines the default retention period, if any.

For instructions on setting up a content-addressed storage area, refer to [Setting up content-addressed storage areas, page 256](#). For information about how to save documents to a content-addressed storage area, refer to [Creating SysObjects, page 132](#), in *Content Server Fundamentals*.

Configuration options

There are four configuration options for a content-addressed storage area that you can set when you create the storage area. These options are:

- A retention period
- Content compression
- Content storage as embedded blobs
- Number of retries for write attempts

Content compression is supported only if you have also installed Content Server with the Content Storage Services license.

Retention periods

Content-addressed storage areas support the definition of a retention period for content stored in those storage areas. The retention period is defined at the storage-area level and is enforced by the Centera host system. If a retention period is defined, the content cannot be removed from the storage system until the period expires even if the associated document is removed from the repository.

Note: The retention period defined for a ca store storage area is independent of the retention policy management features supported by Content Server. However, if the SysObject that contains the content has an associated retention policy, the retention date defined by the policy is also stored as metadata in the Centera system, and the retention date furthest in the future is enforced. For more information about defining retention periods and how retention policies and ca store retention periods interact, refer to [Defining storage area retention requirements, page 257](#).

Compression

Content stored in a content-addressed storage area may be stored as compressed content if you have installed Content Server with a Content Storage Services license. If the storage area is configured to store compressed content, Content Server compresses the content when writing to the storage area and uncompresses the content when retrieving it.

Content compression is enabled when the storage area is created and cannot be reset later.

Whether to link or embed the content in the C-clip

Each piece of content stored in a content-addressed storage area is represented in the storage area by an object called a C-clip. The content address of the content is an XML representation of the C-clip. By default, the actual content is stored in a separate object linked to the C-clip. It is possible to configure a content-addressed storage area to store the content in the C-clip as an embedded blob instead of creating a separate linked object for the content. Embedding the content reduces the number of objects in storage, providing scalability and cost-of-ownership benefits.

When you configure a ca storage area to use embedded blobs, you define the maximum size of the content you want to store as embedded blobs. Any content file that exceeds the specified size is stored in the default way, as a linked object.

The embedded blob configuration option is a pool option. If set in a ca store object, it applies only to content whose `a_storage_type` is set to that ca store object. If there are multiple ca store storage areas that store content in the same Centera cluster, but each has a different maximum size set for embedded blobs, Content Server uses the threshold set in the individual ca store storage area definition to determine whether to embed or link the content in the C-clip.

For instructions on configuring embedded blob use, refer to [Configuring embedded blob use, page 259](#).

This option is changeable after the storage area is created. You can start or stop its use or change the size of embedded blobs.

Configuring write retries

You can configure the retry behavior when content is written to the storage area. You can define the maximum number of times Content Server will attempt to write a content file to the storage area and also whether a final attempt is made after the maximum number of attempts has been made. Those configuration options are controlled by:

- `castore_write_max_attempts` dmcl.ini key
The setting on this key defines the maximum number attempts made by Content Server to write content to the storage area.
- `dm_storage_strategy` set in `a_content_attr_name`
Placing the value 'dm_storage_strategy' in the `a_content_attr_name` (an attribute of the ca store object) notifies Content Server to make one final attempt to write the content to the storage using Full Blob Name mode.

For more information and instructions regarding the retry options, refer to [Configuring write attempts in content-addressed storage areas](#), page 260.

Blob store storage areas

Content stored in blob storage is stored in the repository, in rows in an RDBMS table—one row for each content file. The content must be less than or equal to 64 K.

A blob storage area is implemented using a `dm_blobstore` object, and the name of the resulting table in the RDBMS is derived from the name you define for the blob store object.

Using blob storage can make backing up content easy, because the content is automatically backed up when the repository is backed up. There is also a small performance enhancement when content is stored in blob storage.

You cannot define a blob storage area as the underlying area for a linked store or as a component of a distributed storage area. That is, blob storage cannot be accessed through a linked store storage area or through a distributed storage area.

One attribute, named `ascii`, is defined for the blob store type. It is a Boolean attribute whose value indicates whether the content in blob storage is ASCII strings or arbitrary sequences of 8-bit characters. `TRUE` indicates that the content is in ASCII format. `FALSE` indicates that it is arbitrary sequences of 8-bit characters.

Note: You can store ASCII content in a blob store that has the `ascii` attribute set to `FALSE`. However, you cannot store non-ASCII content in a blob store that has `ascii` set to `TRUE`.

Turbo storage

Content in turbo storage is stored in the repository. Turbo storage is most useful if you are granulating the content of documents; for example, it works well for SGML documents. It also provides enhanced performance for content retrieval.

Turbo storage areas are not represented by storage objects. The content is stored in the `i_contents` attribute of the `dmr_content` object. You can store up to 2000 bytes in the attribute if the RDBMS is Oracle or DB2. If the RDBMS is MS SQL Server or Sybase, you can store up to 255 bytes in the attribute. If the content exceeds those limits, the excess is stored in one or more subcontent objects. Each subcontent object can store an additional 2000 bytes (for Oracle or DB2) or 255 bytes (for MS SQL Server or Sybase).

Content stored in turbo storage must be in ASCII format.

You can use any of the content manipulation methods, such as `Getfile`, `Setfile`, and `Setcontent`, to manipulate content in turbo storage. However, the server cannot automatically generate renditions of content in turbo storage. If you want a rendition of a file in turbo storage, you must create the rendition externally and add it to the repository using the `Addrendition` method.

Distributed storage areas

A distributed storage area does not contain content. Instead, it points to component storage areas that contain the content. The component storage areas of a distributed storage area can be any mixture of file store and linked store storage areas, but all must store the same kind of content (The component storage objects must have the same value in their `media_type` attribute.)

Distributed storage areas are useful when repository users are located in widely separated locations. For example, a company might have offices in New York, San Francisco, Tokyo, and London, with users in each office using the same repository. You can define a distributed storage area with a component in each geographic location and set up the appropriate content replication jobs to ensure that content is current at each location. This provides users in each office with fast access to local copies of the documents.

The *Documentum Distributed Configuration Guide* describes how to implement and administer a distributed storage area. [Distributed Store, page 180](#), in the *EMC Documentum Object Reference Manual* lists the attributes defined for the distributed store object type.

External storage areas

An external storage area represents an external storage device that is known to Content Server but not managed by Content Server. For example, the device might be a CD-ROM, a file system, or an optical device. External storage areas are best used to handle legacy content and data. Storing such content in external storage removes the necessity for importing the content into the repository.

Content Server communicates with the storage device through a user-defined plug-in library. When a user or application issues a request to save or retrieve content in an external storage area, Content Server invokes the plug-in to perform the requested operations. To identify the content, the plug-in and Content Server use a token. The token can be a file path, a URL, or user-defined.

Use constraints

Using external storage has the following functional constraints:

- You cannot issue `setContent`, `insertContent`, or `appendContent` (or their DMCL equivalents) on objects in external storage.
- You cannot replicate objects in external storage.
- You cannot dump with content or load with content.
- The `dmclean` utility drops only the `dmr_content` object and does not delete the physical content file.
- The `dmfilesan` utility does not affect content stored externally.
- You cannot access Macintosh files stored in an external storage area.

Types of external storage areas

There are three types of external storage areas. All are subtypes of the `dm_extern_store` object type. The `extern_store` type is a subtype of `dm_store` and its attributes are those that are common to all external storage areas. The three subtypes of `dm_extern_store` are:

- `dm_extern_file`
- `dm_extern_url`
- `dm_extern_free`

Each external storage area subtype represents one kind of token used to retrieve the content in that area.

Use an external file storage area if the content in the external storage will be represented by a content token in the form of a file path. Typically, this content is legacy files on external file systems, optical disks, or CD-ROMs.

Use an external URL storage area if the content is accessed using a content token in the form of a URL. The URLs must conform to the URL standard. The client and the server do not validate the format of the URL.

Use an external free store storage area if the content is accessed by a user-defined content token that is not in the form of a file path or URL. An external free store lets you define your own token standard and means of retrieving the content associated with the token.

You can create renditions of content stored in an external file store if the server-side plug-in is configured and `rend_backing_store` is specified in the server config object. You cannot create renditions for content stored in external URL or external free storage areas.

Plug-in objects for external storage

Content Server invokes a user-defined shared library or DLL to handle content stored in an external storage device. The shared library or DLL is represented in the repository by a `dm_plugin` object. The shared library or DLL is stored as the content of the plugin object, in a file store storage area. The content's format must be set to a specified format for each platform. (The required format object for each platform is created when Content Server is installed. When you create an external storage area and configure the plug-in, the format is set automatically.)

You can configure the system to execute the plug-in on either the client or the server host.

For better performance, configure the plug-in to be run on the client side if a client-side plug-in is available.

Documentum provides a sample plug-in for the external file store type in the `%DM_HOME%\unsupported\plugins` (`$DM_HOME/unsupported/plugins`) directory.

The API interface between the shared library or DLL and the server consists of C functions for the plug-in library. The functions are described in detail in [Chapter 3, Functions for Creating Plug-in Libraries](#), in the *Content Server API Reference Manual*.

Linked store storage areas

A linked store storage area does not contain any content. Instead, on Windows, it points to a logical link to the actual storage area and, on UNIX, it contains the logical link to the actual storage area. The actual storage area is a file store storage area. ([Linked Store](#),

[page 286](#), in the *EMC Documentum Object Reference Manual* lists the attributes defined for the linked store object type.)

On Windows platforms, the file store storage area is implemented as a shared directory and the file system on which the underlying storage area resides must be a Windows NTFS file system, not a Windows FAT file system.

The indirection provided by a linked store can provide tighter security by allowing you to define the file store storage area as .

Using a linked store provides a way to make files public to users who request them through Content Server while still maintaining operating system-level restrictions on the actual storage directory and contained files. This means that you can ensure that users must access the files using Content Server instead of accessing them through the operating system.

You can make the permissions on the actual storage directory and the files inside as restrictive as you like.

When a Documentum user requests a file from the storage area, the following process occurs:

1. The server makes the underlying file publicly readable.
2. The link is established between the link location and the actual file.
3. The server creates a link record object that stores information about that link.
4. When the session ends, if no other session is using the same link, the link and the associated link record object are destroyed, and the file is no longer public.

Summary of storage area configuration options

[Table 7-1, page 235](#), summarizes the configuration options for the storage areas.

Table 7-1. Summary of storage area configuration options

Storage area type	Configuration option					
	Public or private	Encryption	Compression	Non-duplication of content	Digital shredding	Distributed component
File Store	Yes	Yes	Yes	Yes	Yes	Yes
Content-addressed Store	No	No	Yes	No	No	No
Blob Store	No	No	No	No	No	No
Turbo Store	No	No	No	No	No	No
External Store	No	No	No	No	No	No
Linked Store	No*	No*	No*	No*	No*	Yes
Dis-tributed Store	No*	No*	No*	No*	No	No

* The configuration option may be set at the underlying component level, but is not supported if set in the actual `dm_linkedstore` or `dm_distributedstore` object.

Content and full-text indexes

Content in all types of storage areas is indexed. It is possible to turn off content indexing for an individual document or object, but you cannot turn off indexing at the storage area level or indexing of metadata values. For instructions on stopping content indexing, refer to [Turning off content indexing, page 307](#).

Content stored in an encrypted storage area is not encrypted in the index. Similarly, content stored in a compressed storage area is not compressed in the index.

For a complete description of full-text index architecture and how indexing works, refer to [Chapter 8, Full-Text Indexing](#).

How objects, contents, and storage are connected

Content objects are used to associated objects and their content files. The first time the server places a content file in storage, it creates a content object for the file. The content object identifies the SysObject that contains the content file and the storage area in which the content file is stored.

The attribute `parent_id` in the content object contains the object IDs of all objects to which the content file belongs. After the storage area is determined, it is recorded in the `storage_id` attribute of the content object and in the `a_storage_type` attribute of the associated document object. The `storage_id` attribute contains the object ID of the storage area. The `storage_id` value reflects the storage area defined for the first document to which the content is added. The content file is stored in that area even if you later bind the file to other documents for which a different storage area is defined.

The storage area's storage object has an attribute that points, directly or indirectly, to an actual storage area:

- If the storage type is file store, the attribute is called `location_name`. The `location_name` attribute contains the name of the location object that contains the full directory path of the storage area.
- If the storage type is linked store, the attribute is called `link_location`.

On Windows platforms, the `link_location` attribute points to the location object that references the mount point object representing the Windows shared directory that is the linked storage area's underlying file store storage area

On UNIX platforms, the attribute identifies the directory containing the logical link to the actual storage directory.

- If the storage type is distributed store, the attribute is called `r_component`. The `r_component` attribute is a repeating attribute that contains the object IDs of the storage objects for the component storage areas.
- If the storage type is blob store, the attribute is the name attribute of the blob store object. The system creates a table in the repository that has the same name as the blob store object.
- If the storage type is turbo storage, the `a_storage_type` attribute contains the value `dm_turbo_store` instead of the object ID for a storage object.
- If the storage type is content-addressed storage, the first index position [0] in the `a_storage_params` attribute is used to store the IP address of the storage system. Content Server passes this value to the plug-in, which uses it to connect to the storage system.

Allocating content to storage areas

The same administration guidelines that determine what storage areas are created for a site typically determine what content is stored in each storage area. Although users can designate a particular storage area for a document's content by setting the `a_storage_type` attribute explicitly, generally, content is stored in accordance with business rules defined by system administrators.

The rules governing content assignment can be set up using content assignment policies or by setting the appropriate attribute at the type or format object level and using the default storage algorithm. [Using content assignment policies, page 237](#) describes the implementation of content assignment policies and the requirements for using them. [Using the default storage algorithm, page 241](#), describes how to use the default storage algorithm.

Using content assignment policies

The use of content assignment policies is supported only if you have installed Content Server with a Content Storage Services license.

What content assignment policies are

Content assignment policies are a representation of the business rules that govern where content is stored in an enterprise.

Content assignment policies define a content's storage area based on rules. The rules are conditions based on content size or format. For example, a rule might be `content_size>10,000 (bytes)` or `format='gif'`. A rule can have up to five conditions ANDed together. You can also define a custom rule. The rules in a content assignment policy are stored as the policy's content. Documentum Administrator's online help or User Guide provides information about creating rules based on document properties. If you need assistance in creating, implementing, or debugging a custom rule, you must contact Documentum Professional Services or Documentum Developer support.

Each rule in a policy is identified with a particular storage area. The storage area can be either a file store storage area or a ca store storage area. When a policy is applied to a document, the document is tested against each rule in the policy and when a rule is satisfied, the content is stored in the storage area identified with that rule and the remaining rules are ignored.

Creating content assignment policies

Content assignment policies are defined in Documentum Administrator. You must have installed Content Server with a Content Storage Services license to access the links and pages that allow you to define a content assignment policy.

If the Docbase has such a license, Documentum Administrator allows users with at least Sysadmin user privileges to define a policy based on document properties or a custom rule. Documentum Administrator's online help or User Guide provides information about creating rules based on keywords or system-defined SysObject attributes.

If you need assistance in creating, implementing or debugging a custom predicate rule, you must contact Documentum Professional Services or Documentum Developer support.

The target storage areas of the rules you define in a policy can be either file store storage areas or distributed store storage areas.

Assignment policies are stored in a subfolder in the System cabinet. All users in the Docbase (dm_world) must have at least Read access to the policies.

Enforcement of assignment policies

Content assignment policies are enforced by a internal facility of the DFC (Documentum Foundation Classes) called the policy engine. Any client application built on the DFC (5.2.5 SP2 and higher) enforces content assignment policies automatically if the repository is enabled for assignment policies.

Content assignment policies are enforced:

- When new content is created and saved or imported to the repository, whether the content is a primary content file or a rendition unless you explicitly identify a storage location for the content.
- When an application issues a Checkin method against an object.

The DFC IDfSysObject's checkin method automatically issues a setfile method against the object, which invokes the policy engine, regardless of whether the content is changed or not. (For Documentum clients, this means that the policy engine is invoked whether the object is checked in as a new or the same version.) The policy is applied to the new copy of the content file.

Assignment policies are not enforced if:

- You simply fetch an object, change its attributes, and save the changes.
- An application sets the object's a_storage_type attribute.

If `a_storage_type` is set to explicitly identify a storage area for the primary content, then the policy engine does not apply an assignment policy to the primary content. (Note that Documentum client applications do not typically set `a_storage_type`.)

- Similarly, if a storage area is specified explicitly in the arguments to an `Addrendition` method, the policy engine does not apply an assignment policy to the rendition that is being added.
- The assignment policy for the particular object type is inactive and there are no policies or no active policies among the type's supertypes
- The DFC policy engine is turned off
- An assignment policy does not exist for the particular object type or for any of the type's supertypes
- The document does not satisfy any of the conditions in the applicable policy
- The content is associated with an object that is a replica, is added to the repository through a dump and load operation, or is generated by a `Refresh` method.
- The content is saved with a retention date.

DFC assignment policy information cache

The DFC maintains a cache of assignment policy information. The cache is updated at intervals defined by the `dfc.storagepolicy.validation.interval` property in the `dfc.properties` file. The default interval is 5 seconds. You can reset the interval.

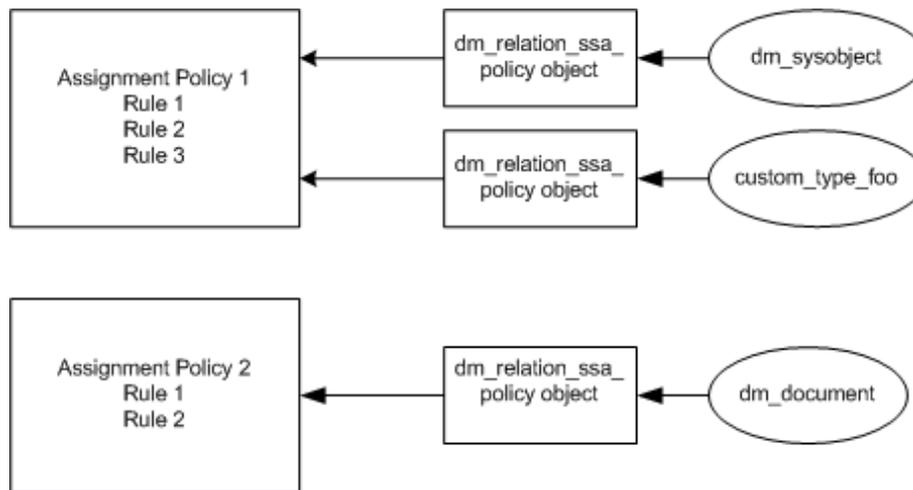
If you increase the interval, then it takes longer for policy changes to be recognized by the policy engine. If you decrease the interval, changes may be available for use sooner but performance may be degraded.

For information about the location of the `dfc.properties` file and how to set properties, refer to the *DFC Release Notes for 5.3*.

Internal implementation of assignment policies

Content assignment policies are stored in the repository as `dm_ssa_policy` objects and related to a `SysObject` object type by an object of type `dm_relation_ssa_policy`. The `dm_relation_ssa_policy` type is a subtype of `dm_relation`. An `relation_ssa_policy` object records the object ID of the `ssa_policy` and an object type to which the policy applies.

A single content assignment policy (represented by an `ssa_policy` object), can be related to multiple object types. However, a single `SysObject` object type can be related to only one content assignment policy. [Figure 7-1, page 240](#), illustrates how content assignment policies, as represented by `ssa_policy` objects and object types are related.

Figure 7-1. Relationship of content assignment policies and object types

Note: If you delete an object type that has a content assignment policy, the relation ssa policy object that associates the type with the policy is automatically deleted also. The ssa policy object is not deleted automatically, as it may be related to other object types. If you want to delete it, you must do so manually.

Algorithm used by the DFC policy engine

The DFC policy engine uses the following algorithm to determine the storage area for content files:

1. If the `a_storage_type` attribute of the object is explicitly set before the document is saved or a storage area is explicitly specified in an `Addrendition` method, put the content in the specified storage area.
2. If a storage area is not explicitly specified, look for an assignment policy for the object type of the document.
3. If a policy is found, test the document against the rules in the policy. When a rule matches, store the content in the storage area identified for that rule. If no rule matches, use the default storage algorithm to determine where to store the content. (The *Content Server Administrator's Guide* describes the default storage algorithm.)
4. If a policy is not found for the object type, traverse up the object's supertypes, to determine if any have an assignment policy. If a policy is found, test the document against the rules in that policy. If a rule matches, store the content in the storage area identified for the rule. If no rule matches, use the default storage algorithm to determine where to store the content. (The *Content Server Administrator's Guide* describes the default storage algorithm.)

5. If no policy is found in the object's type tree, use the default storage algorithm.

Assignment policy administration

There are a variety of administration-related operations that you can perform for content assignment policies. For example, you can turn on tracing of the policies. You can also disable the use of a particular policy, turn off use of all policies, and turn off error reporting for individual policies. For instructions on these procedures, refer to [Administering content assignment policies, page 279](#).

Using the default storage algorithm

The default storage algorithm uses values in a document's object, its associated format object, or type definition to determine where to assign the content for storage. The algorithm is different for primary content and renditions.

The default storage algorithm is used when

- Storage policies are not enabled
- Storage policies are enabled but a policy does not exist for an object type or for any of the type's supertypes
- A content file does not satisfy any of the conditions in the applicable policy
- Content is saved with a retention date.

Primary content algorithm

The default storage algorithm for primary content is:

1. If `a_storage_type` is set for the object, store the content in the storage area specified in that attribute.

A user or application can specifically set the object's `a_storage_type` attribute before saving the object, to assign the content file to a storage area.

2. If `a_storage_type` is not set, check for a value in the `default_storage` attribute of the content's associated format object. If that attribute is set, store the content in the specified storage area.

Note: Only the `jpeg_th` and `jpeg_story` formats have `default_storage` set by default. For those formats, the attribute is set to the thumbnail storage area. For all other

formats, the attribute must be manually set if you want to store content in a particular format in a particular storage area.

3. If neither `a_storage_type` nor the format's `default_storage` is set, store the content in the default storage area for the object type.

The SysObject type and every subtype has a default storage area for storing content files associated with objects of the type. The storage area is defined in the `default_storage` attribute of the type's type info object. During Content Server installation, the default storage area for the SysObject type is set to the file store storage area created during the installation (`filestore_01`). The subtypes of SysObject inherit this default. (However, this can be changed, so `filestore_01` may not be the default at your site or for a particular SysObject subtype.)

4. If none of the above are defined, store the content in turbo storage.

This algorithm examines three attributes:

- `a_storage_type` for `dm_sysobject`
- `default_storage` for the object's format
- `default_storage` for the object's object type

The `a_storage_type` attribute is defined for `dm_sysobject` and inherited by all its subtypes. Typically, `a_storage_type` is set if you want to store an object of a particular type in a location that is different from the default area for its content format or object type. You must set this attribute for an object before you save the object for the first time.

The `default_storage` attribute of a format object identifies a storage area by the storage area's object ID. By default, only the `jpeg_th` and `jpeg_story` format objects have a value in the `default_storage` attribute. For those formats, the attribute is set to the object ID of the thumbnail storage area. (You must have Documentum Media Transformation Services installed to create and use thumbnails.) If you want to store content files in locations specific to the file formats, set the `default_storage` attribute of the associated format objects before saving objects with content files in those formats.

The default storage area for an object type is defined in the `default_storage` attribute of the `dmi_type_info` object for the object type. The Content Server installation procedure sets the `default_storage` attribute for the SysObject type to the `filestore_01` storage area. The value is inherited by all SysObject subtypes. You can change the default using the DQL ALTER TYPE statement.

By setting (or not setting) these three attributes appropriately, you can direct content to desired storage areas when using the default storage algorithm. For example, suppose you want all `dm_documents` except those in XML format to go to `Storage_1` and you want the XML documents to go to `Storage_2`. To obtain this behavior, you set the default storage for the `dm_document` object type to `Storage_1` and default storage for the XML format to `Storage_2`.

Rendition algorithm

When the content file is a rendition of a primary content file, the following algorithm is used to determine where to store the file:

1. If a storage area is defined in the Addrendition method, the rendition's content file is stored in that location.
2. If a storage area isn't defined in the Addrendition method, the server checks for a value in the default_storage attribute of the content file's associated format object.
3. If there isn't a default storage area defined in the format object, the rendition is stored in the same storage area as the object's primary content.

Content retention

Businesses often must retain documents for a specified period of time. Content Server supports two ways to ensure that documents, and their content, cannot be deleted until a specified retention period has expired:

- You can associate the document with a retention policy.

Retention policies are defined and applied using Documentum Administrator. They may be applied to documents stored in any type of storage area. Using retention policies requires a Retention Policy Services license. For more information about retention policies, refer to [Document retention and deletion, page 128](#), in *Content Server Fundamentals* or the *Documentum Administrator's Users Guide*. For information about defining a retention policy, refer to Documentum Administrator online help.

- You can store the document in a content-addressed storage area that has a defined or required retention period.

Documents whose content is stored in a content-addressed storage area cannot be deleted until the retention period expires unless the user has privileges to perform a forced deletion.

For information about defining a retention period for a content-addressed storage area, refer to [Defining storage area retention requirements, page 257](#). For information about forced deletions, refer to [Forced deletions, page 130](#), in *Content Server Fundamentals*. [Enabling forced deletion in content-addressed storage areas, page 268](#), describes how to enable forced deletions for a content-addressed storage area.

This option requires a Content Services for EMC Centera license.

System-defined storage areas

Installing Content Server creates the following default file store storage areas for content:

- `filestore_01`

This is the default storage area for all SysObjects except those that have thumbnail or streaming files as content. Its `media_type` attribute is set to 0.

- `thumbnail_storage_01`

This is the default storage area for objects with primary content in thumbnail format. Its `media_type` attribute is set to 1.

- `streaming_storage_01`

This is the default storage area for object with primary content in streaming format. Its `media_type` attribute is set to 2.

- `replicate_temp_store`

This is where a dump file is stored in the target repository during a load operation. The file is removed after the load operation is completed.

- `replica_filestorage_01`

This is the default storage area for the content associated with object replicas in a repository.

The storage areas are created by the `headstart.ebs` script that runs during the installation procedure. By default, the storage areas are created under `%DOCUMENTUM%\data\<docbase_name> ($DOCUMENTUM/data/<docbase_name>).`

File paths and URLs for content files in storage

This section describes how to interpret a directory path or URL for content files in a file store storage area.

Path specifications for content in file stores

When you store a file in a file store storage area, the server creates a path specification for the file using the following format

On Windows platforms:

storage_area_path\docbase_id\8a\xx\yy\mm[.ext]

On UNIX platforms:

storage_area_path/docbase_id/8a/xx/yy/nn[.ext]

where:

- *storage_area_path* is the path specification in the storage area's associated location object (for example, `dmadmin\data\storage_01` or `u12/dmadmin/data/storage_01`).
- *docbase_id* is the hexadecimal representation of the ID of the repository that contains the content.

Note: You can find a decimal representation of the repository ID in that server's `server.ini` file. This file is found in `%DOCUMENTUM%\dba\config\docbase_name` (`$DOCUMENTUM/dba/config/docbase_name`).

The directories represented by *8a\xx\yy\nn* (*8a/xx/yy/nn*) are named by converting the content's data ticket value into hexadecimal format and dividing it into pairs. The data ticket value is generated when the file is first stored, and a subdirectory is created and named for each of the first three pairs of numbers (*8a xx yy*) in the ticket. The *xx* directory is a subdirectory of the *8a* directory and the *yy* directory is a subdirectory of the *xx* directory. The file is represented by the final two numbers (*nn*) and is stored in the *yy* subdirectory. The *a* can be any number from 0 to f, and *xx*, *yy*, and *nn* can range from 00 to ff. (The 8 increments after there are approximately 256 million content objects in the repository.)

For example, a data ticket value of -2147483077 becomes 8000023b in hexadecimal. The file path generated for a content with this data ticket would be *storage_area_path\docbase_id\80\00\02\3b* (*storage_area_path/docbase_id/80/00/02/3b*).

The name of the file is 3b.

If the `use_extensions` attribute for the storage area is set to `TRUE` and an extension is defined for the file's format (in its format object), the server appends the appropriate extension (*.ext*) to the file name.

URL specifications for content files

Applications can access content files, particularly thumbnails or streaming content, using URLs. The URLs are obtained using the DQL keyword `THUMBNAIL_URL` or the `GET_FILE_URL` administration method.

URLs contain the following information:

- The base URL defined for the storage area
- A path relative to the storage area identified in the `store` attribute that points to the content file
- A `store` attribute that identifies the storage area containing the content file

If the storage area's `require_ticket` attribute is set to `TRUE`, the URL also contains a ticket that contains an encryption of the path plus a time stamp.

For example:

```
http://myserver.documentum.com:8080/getThumbnail?path=00232803/80/00/01/Ob.jpg  
&store=thumbnail_store_01&ticket=8002DWR670X
```

`http://myserver.documentum.com:8080/getThumbnail?` is the base URL.

`path=00232803/80/00/01/Ob.jpg` specifies the path to the file.

`store=thumbnail_store_01` identifies the storage area.

`ticket=8002DWR670X` is the optional ticket.

Setting up storage

The steps required to implement a storage option depend on what option you choose. For example, setting up blob storage is as easy as creating a blob store object. To set up a file store storage area, you create one file store object and one location object. Turbo storage does not require any setup. You simply set an attribute of the content object.

Documentum Administrator is the preferred way to set up storage areas. You can use DQL, if needed, however.

This section includes the following topics:

- [Distributed storage setup, page 247](#), which briefly describes the implementation of creating distributed storage areas
- [Setting up blob storage, page 247](#), which contains information and instructions for creating a blob storage area
- [Setting up file store storage areas, page 248](#), which contains information and instructions for creating a file store storage area.
- [Linked store setup, page 250](#), which contains information and instructions for creating a linked store storage area.
- [Setting up external storage, page 253](#), which contains information and instructions for creating external storage areas
- [Setting up content-addressed storage areas, page 256](#), which contains information and instructions for setting up a content-addressed storage area
- [Setting up turbo storage, page 261](#), which contains information and instructions for using turbo storage areas

Distributed storage setup

A distributed storage area requires one distributed store object and the location and store objects needed to define all component storage areas. The exact number and type depend on what components you choose.

Because distributed storage areas are generally set up to implement a distributed content architecture for distributed repositories, setting them up is described in detail in [Chapter 3, Implementing Single-Repository Models](#), in the *Documentum Distributed Configuration Guide*.

Setting up blob storage

Content stored in blob storage is stored in the repository. Consequently, when you create a blob storage area, Documentum Administrator creates only a blob store object; a location object is not created.

Blob storage is implemented as tables in the underlying RDBMS. Therefore, the name that you assign to the blob store object for the storage area must conform to the rules that govern type names. Additionally, if the repository is running on DB2 and you create multiple blob store storage areas, the names of the storage areas must be unique among themselves to 16 characters. [Names, page 31](#), in the *Content Server Object Reference Manual* describes the rules that govern type names.

When you create a blob storage area, you must indicate what type of content will be stored in the storage area. You can choose either ASCII content or 8-byte characters. The choice is recorded in the `ascii` attribute of the blob object. If you set the content type to ASCII, the attribute is set to T (TRUE), and you can store only ASCII characters in the storage area. If you set the content type to 8-byte characters, the attribute is set to F (FALSE), and you can store non-ASCII or ASCII content in the storage area.

Using DQL to set up blob storage

To create a blob store object using DQL, use the CREATE...OBJECT statement. The syntax for the CREATE...OBJECT statement to create blob store objects is:

```
CREATE "dm_blobstore" OBJECT
SET "name" = 'object_name',
SET "ascii" = true|false
```

For example:

```
1>create dm_blobstore object
2>set name = 'blob_store_1',
3>set ascii = true
```

4>go

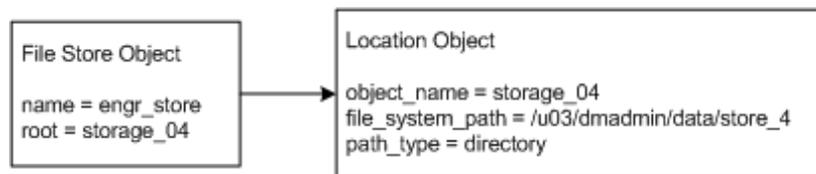
object_name is the name you assign to the blob store object.

Setting up file store storage areas

Creating a file store storage area creates one `dm_filestore` object and one location object. The location object identifies the actual directory location of the storage area. If the location is a shared or mounted directory, one mount point object is created also.

The file store object defines the storage properties of the area. The `root` attribute of the file store object points to the location object that identifies the storage area's location. For example, if you created a storage area named `enr_store` located in `D:\dctm\data\mydb\store_4` (or `/u03/dctm/data/mydb/store_4` in UNIX), you create the file store and location objects illustrated in [Figure 7-2, page 248](#).

Figure 7-2. Relationship between a file store object and a location



Multiple file store objects cannot point to the same location object. Each file store object in a repository must point to a different location object and the location objects must specify different directories.

File extensions and the `use_extensions` attribute

By default, Documentum Administrator checks Use Extensions. This means that the server will add file extensions to file names when it saves a file into a file store storage area. If you do not want the server to add extensions to files saved to the storage area, uncheck the box. You cannot set this after you have begun to put files in a storage area.

The choice is recorded in the `use_extensions` attribute of the area's storage object. This is a Boolean attribute set to `TRUE` if the box is checked and `FALSE` if the box is unchecked.

When the attribute is `TRUE`, only content files whose format definition (in the format object) includes a file extension are affected. If a content file has no extension defined for its format, no extension is set when the file is saved. Refer to [Providing automatic file extensions, page 262](#) for complete details about implementing the use of file extensions when storing and retrieving files.

Setting base URL

Set the Base URL if the storage area will hold thumbnail renditions or content you want to retrieve with a streaming server. A base URL identifies:

- The protocol used for communications
- The host machine on which the server retrieving the file resides
- The port number to use when communicating with the server
- The application root

The format of a base URL is:

```
protocol://machine_id:port_number/server_identification
```

The base URL for storage areas that will store thumbnail renditions is recorded in the installation log file created when the Thumbnail Server was installed. The log file is found in %DM_HOME%\thumbsrv\install\install.log (\$DM_HOME/thumbsrv/install/install.log).

To determine the base URL for storage areas that will store streaming content, consult the documentation provided with the streaming server to obtain the correct base URL value.

The URL you specify is recorded in the base_url attribute and is returned when an application issues a request for an URL.

Defining file store storage areas as public (Windows only)

To define a file store storage area as public, the file system on which the storage area resides must be a Windows NTFS file system, not a Windows FAT file system. Additionally, if any clients are Macintoshes, the Windows server that services the file system must be an Advanced server.

Using DQL to set up file storage

The following procedure outlines how to create a file store storage area using DQL.

To create a file store storage area using DQL:

1. Log in to the repository.
2. Use the CREATE...OBJECT statement to create the location object for the area.

For example:

```
1>create "dm_location" object
2>set "object_name" = 'storage_3',
3>set "file_system_path" = 'd:\documentum\data\store_3',
```

```
4>set "path_type" = 'directory',
3>go

or

1>create "dm_location" object
2>set "object_name" = 'storage_3',
3>set "file_system_path" = '/u12/dmadmin/data/store_3',
4>set "path_type" = 'directory',
3>go
```

For a detailed description of the attributes of location objects, refer to [Location, page 294](#), in the *EMC Documentum Object Reference Manual*.

3. Use the CREATE...OBJECT statement to create the file store object for the area and set its attributes.

In addition to the required attributes, you may also want to set the encryption and compression modes, enable digital shredding, or enable content duplication checking and prevention. Some of these optional features must be set when the storage area is created. Refer to [File store storage areas, page 223](#), for a discussion of these features. For a detailed description of the attributes of file store objects, refer to [File Store, page 245](#), of the *EMC Documentum Object Reference Manual*.

The following example sets only the basic attributes.

```
1>create "dm_filestore" object
2>set "name" = 'enr_store3',
3>set "root" = 'storage_3',
4>set "is_public" = 'false',
5>go
```

The value assigned to the file store's root attribute is the object name of the location object.

Linked store setup

Linked storage area are represented in the repository by two storage objects, two location objects, and one mount point object.

One storage object and location object pair represents the shared or mounted directory containing the link and the other storage object and location object pair represents the directory that actually stores the files. The mount point object is needed because the shared or mounted directory containing the link must be visible to both the client and the server. Consequently, that directory must be mounted by clients and the installation must be using NFS. ([Disabling or enabling the client local area, page 183](#), describes how to enable NFS use for the installation.)

A mount point object has attributes that let you specify the alias for the shared or mounted directory when referenced by clients. Set the appropriate attribute or attributes depending on your site's configuration. Refer to [Chapter 2, Content Repositories](#), for information about creating mount points and setting these alias-related attributes.

Documentum provides a default mount point object whose `file_system_path` defines `%DOCUMENTUM%\share` (`$DOCUMENTUM/share`) as a mounted directory.

The relationships for a linked storage area are shown in [Figure 7-3, page 251](#).

Figure 7-3. The objects that define a linked store storage area



Use file stores as the storage areas that underlie a linked storage area.

You can use Documentum Administrator or DQL to set up a linked storage area. For instructions on using Documentum Administrator, refer to the Documentum Administrator online help.

Using DQL to set up linked storage

The following procedure assumes that the underlying storage area is a file store.

To create a linked store storage area using DQL:

1. Log in to the repository.

2. Use the procedure in [Setting up file store storage areas, page 248](#), to create the file store storage area.
3. Create the location object for the shared directory (Windows) or the directory containing the link (UNIX).

For example, on Windows:

```
1>create "dm_location" object
2>set "object_name" = 'link_05',
3>set "file_system_path" = '\\',
4>set "path_type" = 'directory',
5>set "mount_point" = 'link_mount_05'
6>go
```

For example, on UNIX:

```
1>create "dm_location" object
2>set "object_name" = 'link_05',
3>set "file_system_path" = '/data/link_05',
4>set "path_type" = 'directory',
5>set "mount_point" = 'link_mount_05'
6>go
```

4. Create the linked store object.

For example, on Windows:

```
1>create "dm_linkedstore" object
2>set "name" = 'linkedstore_05',
3>set "component" = 'filestore_05',
4>set "link_location" = 'link_05',
5>go
```

For example, on UNIX:

```
1>create "dm_linkedstore" object
2>set "name" = 'linkedstore_05',
3>set "component" = 'filestore_05',
4>set "link_location" = 'link_05',
5>set symbolic_link = <true/false>,
6>go
```

5. Create a mount point object for the shared directory if the object does not already exist. For example:

For example, on Windows:

```
1>create "dm_mountpoint" object
2>set "name" = 'link_mount_05',
3>set "file_system_path" = '\\dm\dadmin\data\prod\store_05',
4>set "win_preferred_alias" = 'appropriate value'
5>go
```

For example, on UNIX:

```
1>create "dm_mountpoint" object
2>set "name" = 'link_mount_05',
3>set "file_system_path" = '/u12/dmadmin/data/link_dir',
4>set "win_preferred_alias" = 'appropriate value'
5>go
```

Setting up external storage

This section provides an outline of the basic steps in setting up use of an external storage area. EMC Documentum provides standard technical support for creating a plugin object or external storage object. However, for assistance in creating, implementing, or debugging the shared library or DLL that is the content of the plugin object, contact Documentum Professional Services or Documentum Developer Support.

To create an external storage area:

1. Create the shared library or DLL.

The shared library or DLL must implement the functions described in [Chapter 3, Functions for Creating Plug-in Libraries](#), in the *Content Server API Reference Manual*. A sample for the external file store type is provided in the %DM_HOME%\unsupported\plugins (\$DM_HOME/unsupported/plugins) directory. You can use this to guide you when you create your own shared library or DLL.

2. Start Documentum Administrator and connect to the repository.

1. Create a plugin object.
2. Create the external storage area.

It is recommended that you choose to execute the plugin on the server host, as this will ensure that the plugin is compatible with future releases.

Refer to Documentum Administrator online help if needed.

An example of importing documents stored on a CD-ROM

The following procedure outlines how files stored on a CD-ROM can be imported into the repository using an external file store storage area.

To import files on a CD-ROM into the repository using external file store:

This example assumes that the server is running on the Windows platform and the CD-ROM drive letter is E.

1. Create a location object for the CD-ROM drive with `file_system_path` set to E:\.
2. Create the plug-in object.
3. Create a `dm_extern_file` object.
4. Create folders and sub-folders in the repository that resemble the directory structure on the CD-ROM.

5. For each file on the CD-ROM, create a document object under the appropriate folder that uses external file store.

The storage area for the document's content must be the external file store storage area.

6. Use Setpath against each document to set the relative path as the content token.

For example, to create a document object for E:\public\docs\extstore.doc, set the content token in the Setpath command to

```
'public\docs\extstore.doc'.
```

Using the Mount method

The Mount method provides a way to change the path to content in an external file store dynamically. Use the Mount method to dynamically remap the client root location when executing the plug-in on the client if the path name specified in the client root location is invalid or inaccessible.

For example, if you set up external file storage to enable access to files on a CD-ROM and then move the CD-ROM to a local client machine, you can use the Mount command to correct the path. (Refer to [Mount](#), page 312, in the *API Reference Manual* for the syntax.)

In the example in the previous section, if the `a_exec_mode` of the external file store object is set to FALSE (plug-in is executed on the server), you do not need to issue a Mount method. The client can retrieve content using Getfile or Getcontent.

However, if the `a_exec_mode` of the external file store object is set to TRUE but the CD-ROM drive is now F:\, the client must issue a Mount command as follows:

```
mount, c, 6000271280000120, F:\
```

Then, use Getfile or Getcontent to retrieve the content from the CD-ROM.

If the `a_exec_mode` attribute of the external file store object is set to FALSE but a content server is configured on another machine on which the CD-ROM drive is F:, then specify an `a_config_name` and `a_location` pair for the storage object before accessing the content. Set the `file_system_path` of the location object to F:\.

If the `a_exec_mode` attribute of the external file store object is set to TRUE (plug-in is executed on the client), the client application can issue the Mount method in the following format:

```
mount, c, 6000271280000120, local_CD-ROM_drive
```

Then, use the Getfile or Getcontent method to retrieve the content from the CD-ROM.

If the client application does not execute the Mount method but issues Getfile or Getcontent, the content is retrieved on the server side and sent back to the client application if a server side plug-in is available.

External URL storage setup

For objects that use `dm_extern_url`, the `Setfile` and `Setpath` methods require a valid format. The correct format is determined by the type of information the URL points to. [Table 7-2, page 255](#), shows some examples of URLs and formats.

Table 7-2. Examples of URLs and formats

URL	Format
<code>http://www.documentum.com</code>	<code>html</code>
<code>file:///C:/temp/sample.txt</code>	<code>text</code>

Before you set up an external URL store, decide what type of information the URL points to and decide which format to specify in `Setfile` or `Setpath` method.

Use Documentum Administrator to set up external URL storage. For instructions, refer to the Documentum Administrator online help.

Setting up external free storage

The configuration of the plug-in attributes for external free stores is fully under the discretion of users.

Depending on accessibility of the content, users decide whether to run the plug-in on the server or the client.

Content retrieval for objects in external free stores depends on how the user-specified plug-in interprets the content token, so it is up to the user to decide which format to use when using `Setfile` or `Setpath` for objects that use external free stores.

Use Documentum Administrator to set up external free storage. For instructions, refer to the Documentum Administrator online help.

Configuring for optimal performance on retrieval

By default, each time a user in a session issues a `Getfile` method to retrieve content stored in an external storage area, the content is physically retrieved from the storage device and copied to the user's local disk. This default behavior occurs because the content is stored in a storage area that is not managed by Content Server, which means that the content may be changed without Content Server's knowledge. However, if the content is static content that changes infrequently or not at all, you can avoid creating multiple local copies, and possibly enhance `Getfile` performance, by changing the default behavior.

To change the default behavior, set either the `a_content_static` attribute in the storage area object for the external storage area or set the `extern_store_content_static` key in the `dmcl.ini` file. Both the attribute and the key are Booleans. They are F (FALSE) by default, which supports the default Getfile behavior. If you set either to T (TRUE), the default behavior of Getfile changes when a user fetches the same content more than once in a session. Instead of physically fetching the content from the external storage area for second and subsequent accesses, the method returns the file path to the local copy created by the first Getfile on the content.

Setting the `a_content_static` attribute affects all Getfiles that access content in that storage area. Setting the `extern_store_content_static` `dmcl.ini` key affects only Getfiles issued by sessions established using that `dmcl.ini` file.

Setting up content-addressed storage areas

Note: You must have purchased a Content Services for EMC Centera license to create and use a `ca store` storage area. This license is not supported on the HP Itanium platform.

Using content-addressed storage requires the EMC libraries, a plugin object, and a `ca store` object. Installing Content Server automatically installs the EMC libraries and creates the plugin object. The `ca store` object is created when you create a content-addressed storage area using Documentum Administrator.

The libraries are installed in `%DM_HOME%\bin` (`$DM_HOME/bin`). The names differ by platform:

- On Windows, the library is `libemcplugin.dll`
- On Solaris, AIX, and Linux, the library is `libemcplugin.so`
- On HP-UX, the library is `libemcplugin.sl`

The plug-in object is named `CSEC Plugin`. It stores the EMC library as its content. The content is stored in a file store storage area (Storing the plug-in content in a file store storage area is a requirement of the implementation. The storage area may be encrypted.)

Note: The version number of the Centera Runtime libraries and the build version number of the plugin are recorded in the server log file when the server loads the plugin.

The attributes in the `ca store` object identify the storage system metadata fields that you want to set for content stored using that storage object, the location of the storage system, and the plugin object. Optionally, you can also define the retention requirements in the storage object.

Creating a CA storage area does not automatically create an index for the storage area. If you want to index content in a content-addressed storage area, you must create a `fulltext index` object and its associated `location` object for the storage area.

To set up a content-addressed storage area:

1. If the Content Server host machine has multiple versions of EMC Centera SDK installed on it, to ensure that the version installed with Content Server is used by Content Server, make sure that DM_HOME/bin appears first in the appropriate environment variable:

- For Windows hosts, the variable is PATH
- For Solaris and Linux hosts, the variable is LD_LIBRARY_PATH
- For HP-UX hosts, the variable is SHLIB_PATH
- For AIX hosts, the variable is LIBPATH

2. Use Documentum Administrator to create the ca store storage area.

You can define up to 62 values for the Content Attribute Names. The names are recorded in the a_content_attr_name attribute of the associated ca store object. The values defined for a_content_attr_name may not contain spaces.

[Defining storage area retention requirements, page 257](#), describes how to set retention requirements.

[Setting storage parameters, page 259](#), describes how to set storage parameters, including the connection string and embedded blob configuration.

For additional help on setting the storage area attributes, refer to Documentum Administrator online help.

Defining storage area retention requirements

When you create a ca store storage area, you can define a retention period for that storage area. If a ca store storage area has a retention period, you may not remove documents associated with content in that storage area until the period expires unless you use a forced deletion. The retention period is enforced by the Centera host system. (For information about forced deletions, refer to [Forced deletions, page 130](#), in *Content Server Fundamentals*.)

The retention period is stored in association with the content address in the storage system. It is counted down from the date and time at which the content address was generated. For example, if the retention period for DocA is 5 years, and content address for its content was generated on January 1, 2003 00:00:00, the content cannot be removed from the system until January 1, 2008 00:00:00. If you modify the content's metadata in the storage system or perform any other action that creates a new content address, the retention period count down starts again, from the date and time at which the new address was generated. For example, using DocA from the previous example, suppose you issue a Setcontentattrs method against that document's content on February 15, 2006. The method completes and a new content address is generated for DocA's content. This resets the retention period—now the content cannot be removed until February 15, 2011.

Note: If a document is associated with a retention policy and its content is stored in a content-addressed storage area with a retention period, the content's storage-based retention period is set to the longest retention value. For example, if the retention policy mandates keeping the document for five years and the storage area retention period is three years, the document's recorded retention period is five years.

When you create a content-addressed storage area, Documentum Administrator provides two options if you choose to define a retention requirement for the storage area. Whichever option you choose is applied to all content saved to the storage area. The options are:

- You can allow users or the client application to define the retention period when saving content to the storage area.
- You can specify a default retention date to be applied to all content saved to that storage area.

In addition to specifying how the retention period is defined, you must identify which metadata field in the content-addressed storage area will be used to store the retention period. The name of chosen metadata field may not contain spaces.

The decision you make regarding retention requirements is recorded in three attributes:

- `a_retention_attr_required`

The `a_retention_attr_required` attribute defines whether content saved to this storage area requires a retention date. This attribute is set to T if you require users or applications to provide a retention period when saving content.

- `a_default_retention_date`

If you choose to specify a default retention date, the value is recorded in the `a_default_retention_date` attribute.

- `a_retention_attr_name`

The `a_retention_attr_name` attribute records the name of the metadata field that stores the retention period for the content.

It is possible for an application to override a default retention date when executing a `SET_CONTENT_ATTRS` or `Setcontentattrs` method by including the name of the field storing the retention period in the parameter argument of the `Setcontentattrs` method or `SET_CONTENT_ATTRS` administration method. If the retention field name is not included in the parameter argument, the method sets the retention period to the value defined by the `a_default_retention_date` attribute.

To allow users to save content to the storage area without a retention period, do not set any retention requirements when creating the storage area.

Setting storage parameters

The storage parameters provide connection and configuration information to the Centera host.

Defining the connection string

You must define the connection string for the content-addressed storage area. This value is used by Content Server to connect to the Centera host. The connection string is recorded in `a_storage_params[0]`. The basic value is in the format:

```
IP_address|hostname{, IP_address|hostname}?Centera_profile
```

where *IP_address* is the IP address of the Centera host, *hostname* is the host name of the Centera machine, and *Centera_profile* is a full-path specification of a Centera profile. The path must be accessible from the Content Server host machine and the specified directory must be readable by the Content Server installation owner.

Note: The value defined for `a_storage_params[0]` is passed directly to the `FPPool_Open()` Centera SDK function. You can specify any acceptable format of the Centera Connection string as the value. Contact your Centera administrator to determine the correct connection string for your environment.

Configuring embedded blob use

Embedded blobs are a storage option for content stored in content-addressed storage. (Embedded blobs are described in [Whether to link or embed the content in the C-clip, page 229](#).) To configure embedded blob use, enter the following storage parameter value:

```
pool_option:embedded_blob:size_in_KB
```

where *size_in_KB* is the maximum size of the content that you want to store as embedded blobs. For example, if you want to store all content that is 60 KB or smaller as embedded blobs, set the storage parameter value as:

```
pool_option:embedded_blob:60
```

If multiple `pool_options` are defined, Content Server only recognizes the last option. For example, suppose you set `a_storage_params` to:

```
a_storage_params[1]:pool_option:embedded_blob:100
a_storage_params[2]:pool_option:embedded_blob:63
```

Content Server uses 63 as the threshold size for embedded blobs. The setting of '100' is not used or recognized.

There is no maximum set on the size of the embedded blob defined through the `ca store` storage area. However, Centera has a default maximum size for an embedded blob of

100 KB. The default can be overridden by setting a global environment variable in the Centera SDK. If you want to define an embedded blob size larger than 100 KB in the storage area object, you must also override the Centera default by setting the appropriate environment variable in the Centera SDK. (Refer to the Centera documentation for information about that variable.)



Caution: If the maximum size set in the ca store object exceeds the maximum size accepted by Centera, an attempt to write the content to that storage area will fail with an error like the following:

```
[DM_STORAGE_E_CA_STORE_PLUGIN_ERROR_DURING_WRITE]error: "Plugin
ID:<plugin ID> returned error: Wrong parameter detected :
PF_PARAM_ERR, error code:-10006"
```

If the size defined in the storage area object is larger than the maximum size allowed by Centera, the lower of the two values is used to determine whether to store the content as an embedded blob or a linked blob. This also means that if an embedded blob size is not configured in the storage area, the content is always stored as a linked blob even if a threshold is defined in the Centera cluster.

You can configure multiple ca store objects that put content in the same Centera cluster to use different thresholds for embedded blobs. When content is stored, the value defined in the individual ca store object is used to determine whether to store content as embedded blob or as a linked blob.

You can change the threshold after creating the storage area.

Configuring write attempts in content-addressed storage areas

By default, Content Server makes a maximum of three attempts to write content to a content-addressed storage area. You can reset that maximum. You can also configure one last attempt (after the maximum number of attempts are completed) in Full Blob Name mode.

The maximum number of retries is defined in the `castore_write_max_attempts` key in the `dmcl.ini` file. The default value for that key is 3. You can reset the key.

To direct Content Server to make a final attempt using Full Blob Name mode, set the `a_content_attr_name` attribute in the content-addressed storage area's ca store object to `"dm_storage_strategy"`. When this field name is present in `a_content_attr_name`, Content Server will make a final attempt to write content to the content-addressed storage area using Full Blob Name mode.

Setting clocks and time zones for Centera hosts and Content Server hosts

The actual retention date stored in the Centera host for a content file is calculated using the clock on the Centera host machine. Consequently, to ensure calculation of correct retention periods, the time zone information and the internal clocks on Centera host machines and Content Server host machines must be set to matching times (within the context of their respective time zones). For example, if the Content Server host is in California and the Centera host machine is in New York, when Content Server's time is 1:00 p.m. PST, the time on the Centera host should read 4:00 p.m. EST.

Failure to synchronize the times may result in incorrect retention dates for the stored content.

Setting up turbo storage

Content stored in turbo storage is stored in the `i_contents` attribute of the associated content object or in the `i_contents` attribute of a subcontent object if the turbo content is too large to be stored in a content object. The server automatically creates the subcontent objects.

No architectural setup is required to implement turbo storage. To store an object's content in turbo storage, set the `a_storage_type` attribute of the object to `dm_turbo_store` before you issue the `Setfile` method to associate the content with the object. For example:

```
API>create,c,dm_document
. . .
returned_doc_id
API>set,c,returned_doc_id,object_name
SET>mydoc
. . .
OK
API>set,c,returned_doc_id,a_storage_type
SET>dm_turbo_store
. . .
OK
API>setfile,c,returned_doc_id,'c:\harvey\building_proposal'
. . .
OK
API>save,c,returned_doc_id
. . .
OK
```

Providing automatic file extensions

Content Server does not automatically append file extensions to content files when the files are saved to a storage area, copied to the client local area, or copied to a server common area. However, your client applications or systems may require file extensions. If you are using public or linked storage areas, the stored files may also need file extensions so that your applications can access them directly.

You can force the server to append file extensions to content files when they are stored. You can also force the server to append a file extension when the file is copied to a working directory or common area.

Extensions are defined in the objects that define file formats. That is, you define an appropriate extension for a particular format, rather than for a specific file. This is done by setting the `dos_extension` attribute for the format in its format object. If `dos_extension` is set, the server appends that extension to any file of that format that is copied to a client local area or server common area.

If you want the files stored in a storage area to have extensions, you must set the `use_extensions` attribute for the area's storage object when you first define the storage area. This attribute is available for file store and linked store storage areas. It is a Boolean attribute. If it is set to `TRUE`, the server appends the extension defined in `dos_extension` for the file's format object to the internally generated name for the file when it saves the file.

File extension usage must be turned on when the storage area is first defined. You cannot turn it on for a storage area that already contains files without extensions.

It is not an error to save a file without a defined extension to a storage area that uses extensions. However, all files of one format in a storage area must either have extensions or not have extensions. For this reason, you cannot turn on `use_extensions` after you start saving files to a storage area.

Moving content files

Content Server supports three ways to move content files:

- `MIGRATE_CONTENT` administration method

This method allows you to move one content file or multiple content files.

`MIGRATE_CONTENT` method is the recommended way to move multiple content files if you do not have a Content Storage Services license and, consequently, cannot use a content migration policy.

- Content migration policies

Content migration policies are implemented as jobs. The jobs automate the movement of content files from one storage area to another. A content migration policy job invokes the `MIGRATE_CONTENT` method.

You must have installed Content Server with a Content Storage Services license to use content migration policies.

- Records migration jobs

A records migration job moves batches of content files based on criteria you define.

MIGRATE_CONTENT method

`MIGRATE_CONTENT` is the recommended way to move content files. It has a flexible syntax that allows you to move one content file, multiple content files, or all content in one storage area. The syntax also allows you to identify either a content object or a `SysObject` as the target of the operation. If you specify a `SysObject`, you can move the object's first primary file (page 0), renditions of the first page, or both. You can move content files associated with both changeable and immutable objects.

`MIGRATE_CONTENT` lets you move files from a file store, ca store, blobstore or distributed store. You can move the files to a file store, ca store, or distributed store. (You cannot move files to a blobstore.) A file in a retention-enabled content-addressed storage area can only be moved to another retention-enabled content-addressed storage area. There are no restrictions on moving files out of a content-addressed storage area that is not retention-enabled. If you move a file to a distributed storage area, the file is placed in the distributed component local to the Content Server to which you are connected when executing the method.

`MIGRATE_CONTENT` also generates an auditable event called `dm_move_content`. When the event is audited, an audit trail entry is created for each content file that is moved by the method.

You can execute a `MIGRATE_CONTENT` method against an active storage directory. It is not necessary to set the storage area's state to read only to run the method.

For complete information about using this administration method, refer to [MIGRATE_CONTENT](#), page 260 in the *Content Server DQL Reference Manual*. You can execute the method using Documentum Administrator, DQL, or the API. You must be a superuser to execute this method against a content object. To execute the method against the content of a `SysObject`, you must have Write permission on the object.

Content migration policies

Note: You must have installed Content Server with a Content Storage Services license to use content migration policies.

Content migration policies are jobs that automate moving content between storage areas. You can use them to move content from file stores, ca stores, blobstores, and distributed stores to file stores, ca stores, and distributed stores. (Moving content to a blob store is not supported.)

The policies are implemented using a job named `dm_MoveContent` that is installed with Content Server. The `dm_MoveContent` job uses the `MIGRATE_CONTENT` administration method to move content files from one storage area to another. The job selects the content to be moved based on criteria you define when you configure the job. You can select the content based on content size, format, and date. You can also specify a DQL predicate to select the content. The restrictions discussed in [MIGRATE_CONTENT method, page 263](#) on moving content files from retention-enabled content-addressed storage areas apply to content migration policies.

In addition to configuring the default job, you can create additional content migration jobs. Content migration jobs are created and configured using Documentum Administrator. (Use the Migration Policies node.)

Note: You must have installed Content Server with a Content Storage Services license to configure the job or create additional content migration jobs. If you do not have a Content Storage Services license, Documentum Administrator does not display the Migration Policies node nor migration jobs in the list of jobs.

Configurable arguments

When you configure a content migration job, you can configure the job to query against content objects or SysObjects. If the job runs against SysObjects, you can specify whether you want to move primary content, renditions of the primary content, or both. However, note that if a SysObject has multiple primary content pages, only the first primary content (page 0) or its renditions or both are moved.

Additionally, you can specify:

- The storage area to which you want to move the selected content files

The target storage area must be a file store, a ca store, or a distributed store storage area. (If the target is a distributed store storage area, the file is placed in the distributed component local to the Content Server to which the job is connected when executing the method.)

- The selection criteria for the content to be moved
- How many content files you want to move in one execution

- The maximum number of files to move in one execution

For detailed information about configuring this job, refer to the *Documentum Administrator online help* or to the *Documentum Administrator's User's Guide*.

Note: By default, the job removes the migrated content files from the source storage area. This is not a configurable parameter when you migrate content files using the job. Unless the content file is referenced by multiple content objects, the file is removed from the source storage area.

Generated log files

A content migration job generates a log file, like other system-defined jobs, that is stored in `/dba/log/docbase_id/sysadmin`.

Records migration jobs

Records migration jobs are an alternate way to move a large number of content files if you don't want to execute `MIGRATE_CONTENT` manually or if you do not have a Content Storage Services license. Like storage management jobs, records migration jobs are a way to automate storage management rules. For example, you could use a records migration job if you want to move all documents in an lifecycle state called obsolete to an archival storage area.

The target storage area can be another file store storage area or a secondary storage medium, such as an optical juke box or a tape. If the target storage area is secondary storage, the storage must be defined in the repository as a storage area. That is, it must be represented in the repository by some type of storage object.

When you define the records migration job, you can define parameters for selecting the files that are moved. For example, you might want to move all documents that carry a particular version label or all documents created before a particular date. All the parameters you define are AND'ed together to build the query that selects the content files to move. Queries for records migration jobs operate on SysObjects only. Additionally, you cannot audit content moved using a records migration job.

When a records migration job runs, it generates a report that lists the criteria selected for the job, the query built from the criteria, and the files selected for moving. You can execute the job in report-only mode, so that the report is created but the files are not actually moved. The report can be viewed through Documentum Administrator. For instructions, refer to [Reports and trace log files, page 462](#).

Use Documentum Administrator to create Records migration jobs. You must have Superuser privileges to do so.

Auditing content movement

Executing a MIGRATE_CONTENT administration method generates a dm_move_content event.

The dm_move_content event is not audited by default. To start auditing this event, you must issue an Audit method. You can issue the Audit method against a particular content object (representing one content file) or a SysObject, or the object types (dmr_content and dm_sysobject).

If you issue the Audit method against a content object, an audit trail entry for the event is created whenever MIGRATE_CONTENT is used to move the content file represented by the content object, regardless of whether the content is referenced in MIGRATE_CONTENT directly, by the content object ID, or indirectly, through the containing SysObject. If you issue the Audit method against a SysObject, an audit trail entry for the event is generated whenever any content file contained by the SysObject is moved using MIGRATE_CONTENT, regardless whether the administration method references the content directly or indirectly.

For example, suppose there is a document with the object ID of 090000017345a2df and it contains the content file represented by the content object 06000001257a13ce. Suppose also that you issue the following Audit method to audit movement of the document's content:

```
audit,c,090000017345a2df,dm_move_content
```

Later, someone issues the following MIGRATE_CONTENT call:

```
EXECUTE migrate_content FOR 06000001257a13ce  
WITH target_store='filestore_02'
```

The MIGRATE_CONTENT method generates an audit trail entry for the dm_move_content event for the content because the content is contained by the document for which the Audit method was issued.

Maintenance operations for storage areas

This section contains information about and instructions for maintaining storage areas. Some maintenance operations, such as [Moving file store storage areas, page 267](#), typically are not necessary very often. Others, such as [Removing orphaned content objects and files, page 269](#), should be done on a regular basis.

Changing a storage area's state

A storage area can have one of three states: online, offline, or read only.

When a storage area is online, users can store files and copy files out when needed.

When a storage area is offline, users cannot save content in the area or retrieve content from the area. It is sometimes necessary to put storage areas offline. For example, if you want to move a storage area, you must first put the area offline. Or an event, such as a hardware problem, can require you to put a storage area offline.

When a storage area is read only, users can view the content in that area, but if they change a file, the changed file must be stored in a different storage area.

Use Documentum Administrator to change a storage area's state. You can also change the state by executing the `SET_STORAGE_STATE` administration method. You can execute the administration method using a DQL `EXECUTE` statement or an API Apply method. If you want to execute the method directly, refer to [SET_STORAGE_STATE, page 316](#), in the *Content Server DQL Reference Manual* for information about the arguments.

Determining the state of a storage area

You can check the status of a storage area to determine whether it is online or offline through the Storage management pages in Documentum Administrator. Alternatively, you can query the `r_status` attribute directly. Use the following syntax to query the status:

```
SELECT "r_status", "r_object_id" FROM "dm_store"
WHERE "name" = 'storage_area_name'
```

Moving file store storage areas

At times, you may want to move an entire file store storage area. For example, if you reorganize your hardware, you may need to move a storage area to a different disk.

Use the following procedure to move a file store storage area. The procedure describes how to move an entire storage area. It does not describe how to move individual files from one storage area to another.

To move a file store storage area:

1. Log in to the repository.
2. Set the storage area offline.

```
EXECUTE set_storage_state
WITH store = 'filestore_name', offline = TRUE
```

3. Copy the files in the storage area to the new location.

On Windows:

```
c:>copy /s /e source_directory target_directory
```

On UNIX:

```
% cp -r -p source_directory target_directory
```

where *source_directory* is the top-level directory in the current storage area and *target_directory* is the new directory for the storage area. If the target directory does not already exist when this command is issued, the command creates it and copies the files and subdirectories into it from the source directory.

If the target directory does exist when this command is issued, the command copies the source directory (and all files and subdirectories) into the target directory as a subdirectory.

4. Set the `file_system_path` attribute of the `dm_location` object associated with the file store object to point to the new directory for the storage area.

```
UPDATE "dm_location" OBJECT  
SET "file_system_path" = new_directory_path  
WHERE "object_name" = location_object_name
```

Note: In a file store object, the root attribute contains the object name of the location object associated with the storage area.

5. Use the `Reinit` method to reinitialize the server and make the change visible.

```
reinit,c
```

6. Put the storage area back online:

```
EXECUTE set_storage_state  
WITH store = 'filestore_name', offline = FALSE
```

If you have no problems retrieving the contents of documents from the new storage area, and you want to remove the old storage area, you can.

Enabling forced deletion in content-addressed storage areas

Forced deletion allows a user to delete a document whose retention period is unexpired. If the content is stored in a content-addressed storage area, the user performing the forced delete must connect through Documentum Administrator using a Centera profile that grants the privileged delete permission. (For a complete description of forced deletions, refer to [Forced deletions, page 130](#), in *Content Server Fundamentals*.)

To allow a user with the appropriate repository privileges to perform a forced delete on a Centera host, the profile must be available to the Content Server. The server sends the

profile to the Centera host when making the connection for the user. There are two ways to make the profile available for use:

- You can include the full path to the profile file in the connection string specified in a `a_storage_params` attribute of the ca store object.

Instructions for setting `a_storage_params` are found in [Setting storage parameters, page 259](#).

- You can store the profile file in `DOCUMENTUM/dba/config/repository_name/storage_object_name_centera_profile.pea`

where *repository_name* is the name of the repository and *storage_object_name* is the object name of the ca store object.

For example, if the repository is named `MyTest` and the ca store object is named `MyCAStore`, create a profile for the appropriate content-addressed storage system and copy it to `$DOCUMENTUM/dba/config/MyTest/MyCAStore_center_profile.pea`.

Removing orphaned content objects and files

As users work with a repository, they delete unneeded documents or objects. Because destroying a document or other SysObject does not destroy any content files or content objects associated with that document, you should remove the orphaned files and content objects on a regular basis. Documentum provides two automated system administration tools for this purpose. These tools automate two utilities:

- `dmclean`
- `dmfilesan`

An orphaned content object is a content object that is not referenced by any SysObject. (SysObjects have an attribute called `i_contents_id` that contains the object ID of the content object representing their content.) An orphaned content file is a content file that has no associated content object.

The `dmclean` utility scans a repository and finds all orphaned content objects and, optionally, orphaned content files. It generates a script that you can run to remove these objects and files from the repository.

The `dmfilesan` utility scans a specified storage area (or all storage areas) and finds all orphaned content files. It generates a script that you can run to remove these files from the storage area.

Use `dmfilesan` as a backup to `dmclean`. Use `dmclean` regularly, to clean up both content objects and files. The `dmclean` utility has better performance.

For information about using the tools, refer to [Chapter 12, Tools and Tracing](#). This section describes how to use the utilities manually. `dmclean` is the recommended utility.

There are multiple options for executing the `dmclean` and `dmfilesan` utilities:

- Documentum Administrator
- System Administration tools
- DQL EXECUTE statement
- Apply API method
- From the operating system prompt

Executing either utility through EXECUTE, Apply, or the operating system prompt is a two-step process. First, you execute the utility to generate a script, and then you run the script to perform the actual operations. Executing the operation in two parts allows you to check which objects and files will be deleted before the work is actually performed. The scripts are in a format that you can read easily.

You must have Superuser or Sysadmin user privileges to execute these utilities.

Using `dmclean`

You can execute `dmclean` using Documentum Administrator, the DQL EXECUTE statement, the API Apply method, or from the operating system prompt. The syntax varies slightly, depending on how you choose to execute the utility.

By default, `dmclean` operates on content objects representing content stored in any type of storage area except external storage and content-addressed storage. It also removes the associated content files from the storage areas. You can include an argument on the command line to include orphaned content objects representing files in content-addressed storage in its processing. If you include that argument, how it handles the associated orphaned content depends on the retention requirements set for the storage area and the particular content. [Including content in content-addressed storage areas, page 271](#), describes the behavior in detail.

Additionally, `dmclean` also removes orphaned notes (annotations), internal ACLs, and SendToDistribution workflow templates by default. However, if you want it to remove aborted workflows (and the runtime objects associated with the workflow), you must include the `-clean_aborted_wf` argument. `Dmclean` does not remove aborted workflows by default.

[Table 7-3, page 270](#), lists the arguments that you can include to change the defaults.

Table 7-3. `dmclean` arguments

Argument	Description
<code>-no_note</code>	Directs the utility not to remove annotations.
<code>-no_acl</code>	Directs the utility not to remove orphaned ACLs.

Argument	Description
-no_content	Directs the utility not to remove orphaned content objects and files.
-no_wf_templates	Directs the utility not to remove orphaned SendToDistribution templates.
-include_ca_store	Directs the utility to include orphaned content objects representing files stored in content addressable storage. Note: This argument is not supported when dmclean is run through Documentum Administrator.
-clean_aborted_wf	Directs the utility to remove aborted dm_workflows and all related runtime objects.

(If you need syntax help, enter the name of the utility with the -h argument at the operating system prompt.)

The executable that runs dmclean is launched by a method that is created when you install Content Server. By default, the dmclean executable is assumed to be in the same directory as the Content Server executable. If you moved the server executable, modify the method_verb attribute of the method that launches dmclean to use a full path specification to reference the executable. You must be in the %DM_HOME%\bin (\$DM_HOME/bin) directory when you launch the dmclean executable.

Including content in content-addressed storage areas

The dmclean utility does not operate on content stored in content-addressed storage areas by default. If you want to remove orphaned content files from such storage areas, and their associated content objects, you must use the -include_ca_store argument in the utility's command line.

When you include that argument, the utility includes all orphaned content objects whose storage_id identifies a CA store storage area. Removing the content object and content file fails if the storage system does not allow deletions or if the content's retention period has not expired.

Note: To find and remove the repository objects that have expired content, use the RemoveExpiredRetnObjects administration tool. Then use dmclean with the -include_ca_store argument to remove the resulting orphaned content files and content objects. Refer to [Remove Expired Retention Objects, page 517](#), for information about the administration tool.

Running dmclean using a EXECUTE statement

To run dmclean using the EXECUTE statement, use the following syntax:

```
EXECUTE do_method  
WITH method = 'dmclean',  
arguments = 'list of constraining arguments'
```

Dmclean can remove a variety of objects in addition to content files and content objects. These objects include orphaned annotations (note objects), orphaned ACLs, and unused SendToDistributed workflow templates. These objects are removed automatically unless you include an argument that constrains the utility not to remove them. For example, including `-no_note` and `-no_acl` arguments directs dmclean not to remove orphaned annotations and unused ACLs. If you include multiple constraints in the argument list, use a space as a separator.

The utility automatically saves the output to a document that is stored in the Temp cabinet. (The utility ignores the DO_METHOD's SAVE argument. The output is saved to a file even if this argument is set to FALSE.) The output is a generated IAPI script that includes the informational messages generated by the utility.

Running dmclean using an Apply method

You can run dmclean using the Apply API method. The syntax is:

```
apply, session, NULL, DO_METHOD, METHOD, S, dmclean,  
ARGUMENTS, S, 'list of constraining arguments'
```

When you use the Apply method, you must use uppercase for the keywords and the datatype specifications.

Dmclean can remove a variety of objects in addition to content files and content objects. These objects include orphaned annotations (note objects), orphaned ACLs, and unused SendToDistributed workflow templates. These objects are removed automatically unless you include an argument that constrains the utility not to remove them. For example, including `-no_note` and `-no_acl` arguments directs dmclean not to remove orphaned annotations and unused ACLs. If you include multiple constraints in the argument list, use a space as a separator.

The utility automatically saves the output to a document that is stored in the Temp cabinet. (The utility ignores the DO_METHOD's SAVE argument. The output is saved to a file even if this argument is set to FALSE.) The output is a generated IAPI script that includes the informational messages generated by the utility.

Running dmclean from the operating system prompt

To run dmclean from the operating system prompt, use the following syntax:

```
dmclean -docbase_name name -init_file init_file_name [list of constraining arguments]
```

As `dmclean` is executing, it sends its output to standard output. To save the output (the generated script) to a file, redirect the standard output to a file in the command line when you execute `dmclean`.

name is the name of the repository against which to run the utility. *init_file_name* is the name of the `server.ini` file for the repository's server. These two arguments are required. (Refer to [Chapter 3, Servers](#), for information about the server start-up file.)

`Dmclean` can remove a variety of objects in addition to content files and content objects. These objects include orphaned annotations (note objects), orphaned ACLs, and unused `SendToDistributed` workflow templates. These objects are removed automatically unless you include an argument that constrains the utility not to remove them. For example, including `-no_note` and `-no_acl` arguments directs `dmclean` not to remove orphaned annotations and unused ACLs. If you include multiple constraints in the argument list, use a space as a separator.

Executing the `dmclean` script

Executing `dmclean` as a job with the `-clean_now` argument set to `FALSE`, using the EXECUTE statement, the Apply API method, or from the operating system prompt, generates a script. To actually perform the cleaning operations, you must execute the script using the IAPI utility.

To execute the `dmclean` script:

1. Start IAPI.
The `API>` prompt appears.
2. Enter the following command to execute the script:

```
@scriptfile
```

where *scriptfile* is the name of the generated script.

Using `dmfilesan`

Use `dmfilesan` utility to find orphaned content files—content files that have no associated content object. The utility looks for all orphaned files that are older than 24 hours. Executing the utility generates a Bourne shell script that contains commands to remove the orphaned files found by the utility. The utility does not actually remove the files itself. After the utility completes, you must run the script to actually remove the files.

The utility scans one storage area or all storage areas, searching some or all of the area's subdirectories to find the content files that do not have associated content objects. The utility ignores content files that have a content object. Even if the content object is an orphaned content object (not referenced by any SysObjects), dmfilesan ignores the content file if it has a content object in the repository.

Use dmfilesan as a backup to dmclean. You can execute dmfilesan using Documentum Administrator, the DQL EXECUTE statement, the API Apply method, or from the operating system prompt. On Windows, the utility generates a batch file (a .bat script). On UNIX, the utility generates a Bourne shell script. The executing syntax and the destination of the output vary, depending on how you choose to execute the utility.

If you need syntax help, enter the name of the utility with the -h argument at the operating system prompt.

The executable that runs dmfilesan is launched by a method that is created when you install Content Server. By default, the dmfilesan executable is assumed to be in the same directory as the Content Server executable. If you moved the server executable, modify the method_verb attribute of the method that launches dmfilesan to use a full path specification to reference the executable. You must be in the %DM_HOME%\bin (\$DM_HOME/bin) directory when you launch the dmfilesan executable.

dmfilesan arguments

Table 7-4, page 274, lists the arguments to the dmfilesan utility.

Table 7-4. Arguments to the dmfilesan utility

Argument	Meaning
-s <i>storage_area_name</i>	The name of the storage object that represents the storage area you are cleaning up. If you do not specify this argument, the utility operates on all storage areas in the repository that are not defined as far for the server.
-from <i>directory1</i>	Subdirectory in the storage area at which to begin scanning
-to <i>directory2</i>	Subdirectory in the storage area at which to end scanning

Argument	Meaning
<code>-force_delete</code> <i>Boolean</i>	Controls whether the utility deletes orphaned files that are younger than 24 hours. The default is F, meaning do not delete files younger than 24 hours or less.
<code>-no_index_creation</code> <i>Boolean</i>	<p>Controls whether dmfilesan creates and destroys the indexes on <code>dmr_content.data_ticket</code> and <code>dmr_content.other_ticket</code> or assumes they exist.</p> <p>T (TRUE) means that the utility assumes that the indexes exist prior to the start of the utility. F (FALSE) means the utility will create these indexes on startup and destroy them at the finish. The default is F.</p> <p>Refer to Using the <code>-no_index_creation</code> argument, page 276 for details of use.</p>

Identifying the subdirectories of the scanned storage areas

The `-from` and `-to` arguments identify the storage area subdirectories you want to scan. The values for these arguments are the hexadecimal representations of the repository IDs used to identify subdirectories of a storage area.

The top-level directory of a file store storage area is divided into subdirectories for each repository in the installation. All content files from a particular repository stored in that storage area are stored in subdirectories under the directory named for the repository. For example, suppose the storage area is `d:\documentum\data\storage_1 (/u12/dmadmin/data/storage_1)` and the repository ID is `000023e5`. Any content files from repository `000023e5` stored in `storage_1` are stored in subdirectories of `d:\documentum\data\storage_1\000023e5 (/u12/dmadmin/data/storage_1/000023e5)`

The values for the `-from` and `-to` arguments are the hexadecimal representations of the repository IDs that name the subdirectories. The utility scans all directories that fall numerically between the directories you specify in the `-from` and `-to` arguments.

If you include only the `-from` argument, the utility starts at the specified directory and scans all the directories below it. If you include only the `-to` argument, the utility starts at the top directory and scans down to that directory. If neither are specified, the utility scans all subdirectories of the repository subdirectory. For a detailed explanation of subdirectory numbering, refer to [Path specifications for content in file stores](#), page 244.

Using the `-no_index_creation` argument

The `dmfilesan` utility uses two indexes in its processing. These are indexes on `dmr_content_s.data_ticket` and `dmr_content_s.other_ticket`. By default, the utility creates the indexes when it starts and destroys them when it completes executing. However, in some circumstances, `dmfilesan` cannot obtain enough database resources to create the resources. If the utility cannot create the indexes, the utility fails with an error. This can occur in very busy or very large environments.

To avoid this issue, you can create the indexes manually and run the utility with the `-no_index_creation` argument set to T (TRUE). Use `MAKE_INDEX` to create the indexes.

Using the `-force_delete` argument

By default, the utility scans only for orphaned files that are over 24 hours old. To remove content files that are younger than 24 hours, set the `-force_delete` argument to T (TRUE). This argument is F (FALSE) by default. Setting this argument to T is only recommended if you must run `dmfilesan` to clear disk space. You can also use it to remove a dump file if a load operation fails and leaves behind the temporary dump file in the target repository directory. However, do not run `dmfilesan` with `-force_delete` set to T when there are any other processes or sessions creating objects in the repository.

Running `dmfilesan` using an EXECUTE statement

To run `dmfilesan` using the EXECUTE statement, use the following syntax:

```
EXECUTE do_method WITH method = 'dmfilesan',  
  [arguments = '[-s storage_area_name] [-from directory1]  
  [-to directory2]']
```

The utility automatically saves the output to a document that is stored in the Temp cabinet. (The utility ignores the `DO_METHOD`'s `SAVE` argument. The output is saved to a file even if this argument is set to FALSE.) The output is a generated script that includes the informational messages generated by the utility.

Running `dmfilesan` using an Apply method

You can run the `dmfilesan` utility using the Apply method. The syntax is:

```
apply, session, NULL, DO_METHOD, METHOD, S, dmfilesan [ARGUMENTS, S,  
  "[-s storage_area_name] [-from directory1] [-to directory2]"]
```

When you use the Apply method, you must use uppercase for the keywords and the datatype specifications.

The utility automatically saves the output to a document that is stored in the Temp cabinet. (The utility ignores the DO_METHOD's SAVE argument. The output is saved to a file even if this argument is set to FALSE.) The output is a generated script that includes the informational messages generated by the utility.

Running dmfilesan from the operating system prompt

To run the utility from the operating system, use the following syntax:

```
dmfilesan -docbase_name name -init_file init_file_name
[-s storage_area_name] [-from directory1] [-to directory2]
```

As dmfilesan executes, it sends its output to standard output. If you want to save the output, which is the generated script, to a file, you must redirect the standard output to a file on the command line when you run the utility.

The two arguments, `-docbase_name` and `-init_file`, are required. The *name* is the name of the repository that contains the storage area or areas you are cleaning up. The *init_file_name* is the name of Content Server's server.ini file. This is one of the server start up files. (For more information about these, refer to [Chapter 3, Servers.](#))

The generated script

Executing dmfilesan generates a script. The script comprises a series of remove commands that remove orphaned files found by the utility. For each file, the script lists its data ticket and storage ID. The script also contains a template DQL SELECT statement that you can use in conjunction with the data ticket and storage ID values to check the findings of the utility. Here is a sample of the script (generated on a Windows host):

```
rem Documentum, Inc.
rem
rem This script is generated by dmfilesan for later verification
rem and/or clean-up. This script is in trace mode by default. To
rem turn off trace mode, remove '-x' in the first line.
rem
rem To see if there are any content objects referencing a this file
rem listed below, use the following query in IDQL:
rem
rem c:> idql <docbase> -U<user> -P<pwd>
rem 1> select r_object_id from dmr_content
rem 2> where storage_id = '<storage_id>' and data_ticket =
<data_ticket>
rem 3> go
rem
rem If there are no rows returned, then this is an orphan file.
rem
rem storage_id = '280003c980000100' and data_ticket = -2147482481
del \dm\dmadmin\data\testora\content_storage_01\000003c9\80\00\04\8f
rem storage_id = '280003c980000100' and data_ticket = -2147482421
```

```
del \dm\dmadmin\data\testora\content_storage_01\000003c9\80\00\04\cb
rem storage_id = '280003c980000100' and data_ticket = -2147482211
del \dm\dmadmin\data\testora\content_storage_01\000003c9\80\00\05\9d
. . .
```

Executing the dmfilesan script

Run the dmfilesan script from the operating system prompt.

On Windows:

```
c:> script_file_name
```

On UNIX:

```
% script_file_name
```

You may have to give yourself permission to execute this file. On Windows, do so using File Manager to add appropriate permission to your user account. On UNIX, use the following command:

```
% chmod ugo+x script_file_name
```

Replacing a full distributed storage component

Use this procedure to replace a distributed store component whose disk space has filled up.

To replace a distributed store component:

1. Set the component storage area that resides on the full disk to the read only state. You can use Documentum Administrator to set the component to read only, or you can execute the following DQL statement:

```
EXECUTE SET_STORAGE_STATE WITH STORE = 'component_name',
READONLY=true
```
2. Connect to the server site with the full disk and create a new file store storage area on a disk that has space available.
3. Add the new file store storage area as a component to the existing distributed storage area in the repository.
4. Add the new component to the list of far storage areas in the server config objects for all the servers in the repository except the server that is local to the component.
5. Reinitialize all the server config objects.

Resolving a compromised file store key

A file store key is the encryption key used to encrypt the content files in an encrypted file store storage area. Each encrypted file store storage area has its own file store key.

If the file store key for a particular encrypted storage area is compromised, use the following procedure to resolve the situation.

To resolve a compromised file store key:

1. Create a new encrypted storage area.
2. Migrate the content from the compromised encrypted file store storage area to the new encrypted storage area.
Use the `MIGRATE_CONTENT` administration method to move the content.
3. Delete the compromised storage area.

Administering content assignment policies

This section contains information about managing the features provided with Content Storage Services, a separately licensed feature of Content Server. For information about this feature, refer to [Using content assignment policies, page 237](#).

Tracing policy use

If you want to track the use of content assignment policies, you can turn on logging for policies. The logging feature is turned on at the DFC level. When it is turned on, the policy engine records a message each time a policy is applied. Here is an example of the logged message:

```
20:26:15,991 INFO[Thread-5]com.documentum.fc.client.DfDocument
- Using assignment policy "Content Assignment Policy 5" for
content at page 0 in crtext format belonging to object
Document 5.
```

```
20:26:15,991 INFO [Thread-5]com.documentum.fc.client.
storagepolicy.DfStorageRuleEvaluator_objectID_13 -
Storage rule used to determine store is :
contentInfo.getContentSize()
< 5 && contentInfo.getContentFormat().equals("crtext")
-->strgpoltst_o2.
```

To turn on the logging messages, increase the logging level in the `log4j.properties` file in the DFC installation. Modify the first line in the file to read:

```
log4j.rootCategory=INFO,A1,F1
```

The generated log file is named `log4j.log` and is stored in the installation's logs directory. Typically, this is `C:\Documentum\logs\log4j.log`.

Enabling and disabling assignment policies

When you create a content assignment policy, you must explicitly activate the policy. Whether or not a policy is active is controlled by the `is_activated` attribute in the `dm_ssa_policy` object. To set that attribute, use Documentum Administrator.

If an individual policy is inactive, the policy engine behaves as if no policy exists for the object types that use that policy. When an object with an inactivated policy is saved with new content, the policy engine traverses the object's type tree to find a policy for the object. If it cannot find a policy or no policy rules apply, the default storage algorithm is used.

Turning off the policy engine

The policy engine is turned on by default. Turning it off disables the use of content assignment policies for applications that use that particular DFC installation. To turn off the policy engine for a particular DFC installation, add the following line to the `dfc.properties` file:

```
dfc.diagnostics.storagepolicy=false
```

All DFC-based applications that use that `dfc.properties` file are affected. On Windows platforms, there is only one DFC installation on a host machine, so the scope is any DFC-based application running on that machine. On UNIX platforms, it is possible to have multiple DFC installations. Consequently, on UNIX, you may want to add the specified preference to each `dfc.properties` file present on the host machine.

Archiving and restoring documents

As repositories grow and age, you typically archive older or infrequently accessed documents to free up disk space for newer or more frequently used documents. There will also be occasions when you want to restore an archived document. Documentum provides a mechanism for archiving and restoring documents using the Archive and Restore methods and the Archive tool.

Note: If you want to archive fixed data such as email or check images that will not be changed and must be retained for a fixed period, use the content-addressed storage option, rather than the archiving capabilities described in this section. The content-addressed storage option is capable of storing massive amounts of data with a defined retention period. For information about this option, refer to [Content-addressed storage, page 227](#).

How the process works

Five major components are involved in archive and restore functionality:

- The client requesting the operation
- The repository operator's inbox, where the requests are queued
- The Archive tool, which performs the actual operations
- The archive directory, a staging area for the dump files created by the archiving operations and read by the restoring operation
- The offline storage area, where archived files are moved for permanent storage

When you archive a document, these components interact as follows:

1. The client sends an archive request.
The client can be a user selecting a custom menu item requesting archiving (which issues an Archive method) or an application issuing the Archive method.
2. The Archive method queues a DM_ARCHIVE event in the repository operator's inbox.
3. The Archive tool reads the DM_ARCHIVE event in the inbox and performs the archiving operation.
When the tool is finished, it sends a completion message to the user requesting the operation and to the repository operator.
4. A user-defined DO_METHOD procedure moves the file from the archive directory to permanent, offline storage.

When you restore a document, these components interact as follows:

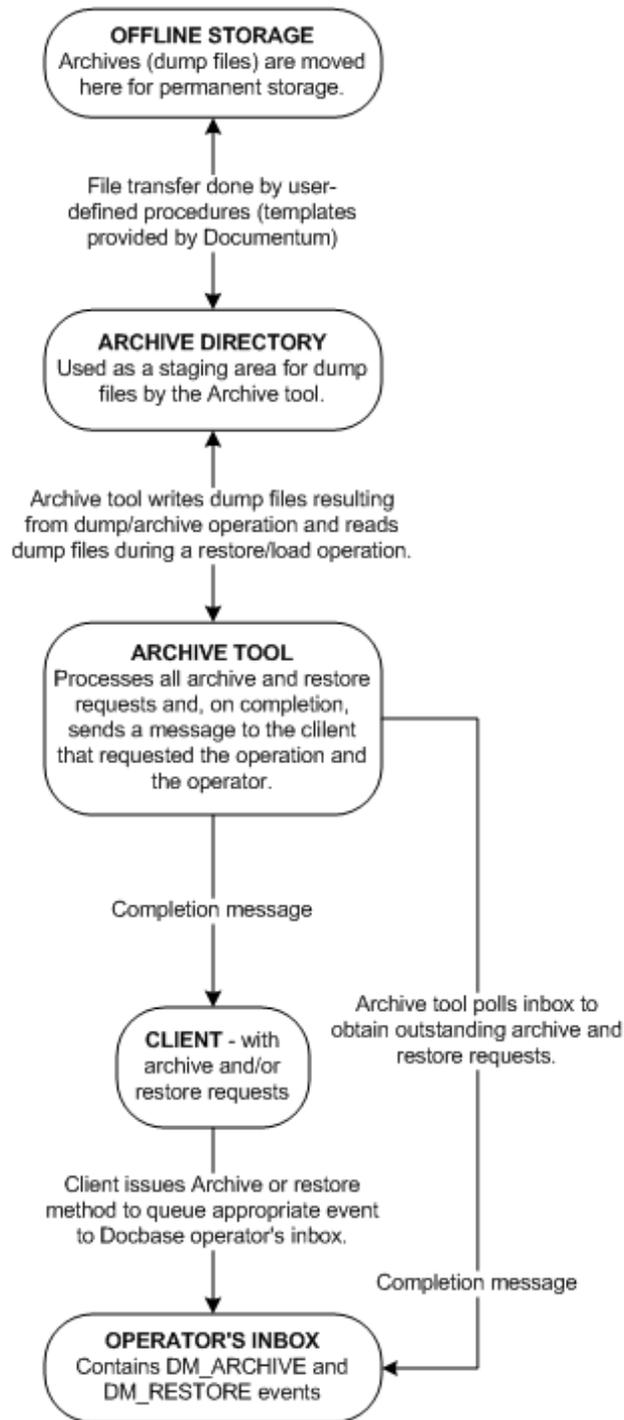
1. The client sends a restore request.
The client can be a user trying to open an archived document using a Documentum client or an application issuing the Restore method.
2. The Restore method queues a DM_RESTORE event to the repository operator's inbox.
3. The Archive tool reads the DM_RESTORE event in the inbox.

4. If necessary the server calls a DO_METHOD function that moves the dump file containing the archived file back into the archive directory.
5. After the file is back in the archive directory, the Archive tool performs the restore operation.

When the tool is finished, it sends a completion message to the user requesting the operation and to the repository operator.

[Figure 7-4, page 283](#), illustrates the archive and restore mechanism. The following sections describe the process in more detail.

Figure 7-4. The archive and restore process



The Archive and Restore methods

Executing an Archive method queues a DM_ARCHIVE event in the inbox of the repository operator, and executing a Restore method queues a DM_RESTORE event in the operator's inbox. The repository operator is a repository user, real or virtual, who is designated to receive all archive and restore requests. The Archive tool, which processes these requests, polls this user's inbox.

For information about the syntax of Archive and Restore, refer to the Javadocs for the DFC methods and to the *Content Server API Reference Manual* for information about the DMCL API methods.

Note: When you issue an Archive or Restore method, do not include the keyword (ALL) with the type name in the predicate. The dmarchive utility called by the Archive tool automatically adds (ALL) to the type name. If you include (ALL) also, the utility fails with a syntax error when it tries to execute the query derived from the predicate.

The return value for both methods is the object ID of the queue item object that represents the queued event.

The Archive tool

The Archive tool reads the queue in the operator's inbox and archives or restores the requested documents. (Refer to [Options for archiving](#), page 287, for more information about this tool.)

Archiving

Each time the Archive tool reads the operator's inbox, it gathers the archiving requests and performs a single dump operation. The resulting dump file contains all documents that had an outstanding archive request.

For each document, the Archive tool:

- Dumps the document and any content files
- Removes the content from the storage area
- Sets the document's a_archive attribute to TRUE
- Sets three attributes of the content's content object:
 - is_offline and is_archived are set to TRUE.
 - set_file is set to the full path of the dump file in the archive directory.
- Sends a message to the user who requested the archive indicating completion

The document object itself is not removed from the repository. Only its content is removed from the storage area. However, if a user tries to open the document (through a client application or using Getfile or Getcontent), the user receives a message that the document is archived and inaccessible.

To perform the dump, the Archive tool generates a dump record object, using the information supplied in the outstanding archive requests. For example, suppose the Archive tool found the following outstanding archive request:

```
archive,c,dm_document where owner_name = 'carolnew'
```

This generates the following DQL statement to create the dump record object:

```
create dm_dump_record object
set file_name = 'ARCHIVE_DIR\filename',
set include_content = true,
append type = dm_document,
append predicate = (owner_name='carolnew' and
r_lock_owner=' ') and a_archive=0,
append type = dmr_content,
append predicate =any parent_id_i in
(select r_object_id_i from dm_document (all)
where (owner_name='carolnew') and r_lock_owner=' ') and
is_archived = 0)
```

Note: file_name is shown in Windows format. On UNIX, it would be ARCHIVE_DIR/filename.

Including a_archive=0 in the qualification ensures that previously archived documents are not archived again.

The dump file's file name is assigned to the set_file attribute of the content objects representing the contents of the archived documents. The file name is generated automatically from the object ID of the dump record object and has the following format:

```
ARCHIVE_DIR/dump_record_object_ID
```

where ARCHIVE_DIR is the path specification of the archive directory.

When the archive operation is finished, the Archive tool queues a notification to the user who requested the archive and the repository operator. The Task Name for this notification is event and the Event value is dm_archive_done.

Because the Queue method requires that an actual object be queued, the Archive tool creates a SysObject named dmarchive_action_complete that is queued with the notification.

The verbose argument on the Archive tool command line directs the tool to print trace messages. Trace messages are messages issued by the Archive tool as it is executing. If you want this functionality, append the verbose argument to the end of the Archive tool command line. For example:

```
dmarchive docbase_name [-Username] -Ppassword
-archive_dir directory -verbose
```

Restoring

Restoring a document loads the document's content from the dump file back into the repository, into its original storage area, and sets the `is_offline` attribute of the content object to `FALSE`. It does not reset the content object's `a_archive` or `set_file` attributes, because although the document has been restored, it also still remains in the archive file.

Before the Archive tool can restore a document, the dump file that contains the document must be present in the archive directory.

The Archive tool creates a load record object to satisfy a restore request. When the load record object is saved, the requested content is loaded back into the original storage area from the dump file. If there are multiple restore requests, the Archive tool groups them as much as possible, to reduce the number of necessary load operations.

For example, suppose the Archive tool finds the following restore request:

```
restore,c,dm_document where owner_name = 'carolnew'
```

The tool generates the following DQL statement to create the necessary load record object:

```
create dm_load_record object
set file_name = 'dump_file_name',
append type = dmr_content,
append predicate = any parent_id_i in
(select r_object_id_i from dm_document (all)
where owner_name = 'carolnew')
and is_archived = 1 and is_offline = 1
and set_file = '%dump_file_name'
```

When the restore operation is finished, the tool queues a notification to the user who requested the restore and the repository operator. The Task Name for this notification is `event` and the Event value is `dm_restore_done`.

Because the Queue method requires that an actual object be queued, the Archive tool creates a SysObject named `dmarchive_action_complete` that is queued with the notification.

Moving dump files on and off line

The archive directory is intended as a staging area. The Archive tool puts dump files generated by archive requests in this directory and reads dump files in this directory when processing restore requests. The archive directory is not usually a permanent storage area for these dump files. The dump files are generally moved to offline storage, such as a tape or optical storage device, and moved back to the archive directory only if they are needed for a restore request.

To make the process of moving files out of and into the archive directory easier, Documentum provides three DO_METHOD procedures. Each procedure is a script that contains comments concerning its use and an example command that is commented out. If you want to use these scripts, edit the procedures. The scripts are found in %DOCUMENTUM%\product*version*\bin (\$DOCUMENTUM/product/*version*/bin). Copy them to %DOCUMENTUM%\dba (\$DOCUMENTUM/dba) and edit the versions in this directory. That way, you will not lose any script customizations if you upgrade your server. The scripts exit with a return value of 0.

The procedures are:

- post_archive

The post_archive procedure moves a dump file to offline storage. The procedure accepts the dump file name as an argument.

- pre_restore

The pre_restore procedure moves a dump file from off-line storage back to the archive directory. The procedure accepts the dump file name as an argument.

- post_restore

The post_restore procedure removes a dump file from the archive directory. Use this procedure only if a copy of the dump file remains in off-line storage.

Archiving content used in multiple documents

If you archive a document that contains a content file that appears in additional documents, the Archive tool does not actually move the content file off line unless all the documents containing that content are archived.

Options for archiving

Because the Archive tool is configurable, you have several options about how to implement archiving at your site.

Be careful of creating extremely large archive files. Each time the utility checks the inbox and archives, it puts all the documents it is archiving in one dump file. This means you must move the entire dump file back to the archive directory to restore a single document in the file. If the files are extremely large, this can be a significant performance hit for restore operations.

Scheduling archiving

You can change the schedule for the Archive tool and run the tool on demand. Refer to [Scheduling jobs, page 148](#), and [Running administration jobs on demand, page 465](#), for more information.

Shortening the run interval may give you a faster response time for requests. However, the archive tool works more efficiently when operations can be executed in groups—that is, when it can operate with as few dumps and loads as possible.

Note: The clock for the polling interval starts when the utility completes all the operations requested.

Types of request

The Archive tool can be configured to process:

- A specific event
- Archive events only
- Restore events only

You can start two instances of the tool and separate the archive and restore operations.

The repository operator

When you set up archiving, decide who to designate as the repository operator. The repository operator is the user whose inbox receives all archive and restore requests. This is where the Archive tool looks for unprocessed archive and restore requests.

Your repository operator may or may not be a real person. You may find it more practical to create a virtual user—a user who exists only within the repository and operating system. This gives you an inbox dedicated to archive and restore requests. This is a good choice if you want to keep an audit trail of archive and restore activity.

If a repository operator is not explicitly named on the Archive tool command line or in the Archive or Request method, the server assumes the operator is the user named in the `operator_name` attribute of its server config object. By default, this attribute is set to the repository owner. Documentum users with Sysadmin or Superuser privileges can change this attribute. Use Documentum Administrator to change the attribute. You must reinitialize the server after changing the attribute.

Moving the dump file in and out of the archive directory

Although you can use the `post_archive` and `pre_restore` procedures to move files in and out of the archive directory, you can also use an alternative automatic archiving product. For example, if the archive directory is on a NetStor device, NetStor manager can take care of moving the files to and from optical storage.

Choosing an archive directory

The archive directory is the disk location where the Archive tool places the dump files it creates when archive requests are processed. It is also the location where the Archive tool expects to find the dump files when it processes restore requests. This location is not expected to be a permanent storage location for the archive file. (Refer to [Moving dump files on and off line](#), page 286, for more information about using this directory.)

When you archive a document, the tool records the location of the archive directory in the `set_file` attribute of the content object that links the document and its content. The `set_file` attribute is set to the full path of the dump file that contains the archived document.

During a restore operation, the tool expects to find the dump file in the directory specified in the `set_file` attribute. Consequently, the disk location that you choose for the archive directory must be:

- Large enough to accommodate all dump files you might reasonably expect to be there at the same time
- Permanent

Implementing archiving

To implement archiving for a repository:

1. Choose a location for the archive directory. Create the directory if necessary. The location should have enough disk space to accommodate all dump files expected to be there at any given time, and it should be permanent.
2. Choose a user to serve as the repository operator, or create a new virtual user to be the repository operator.
3. Set the `operator_name` attribute of the server's server config object to the user name of the user you chose as the repository operator.

```
UPDATE "dm_server_config" OBJECT
SET "operator_name" = 'username'
WHERE "object_name" = 'server_config_name'
```

If there are multiple servers running for the repository, be sure to set this attribute for each server.

4. Identify which requests you want the Archive tool to process.
Use Documentum Administrator to modify the arguments of the dm_DMArchive job to define the requests you want to archive.
 - To archive a specific event, type

```
queue_event event_id
```

where `event_id` is the object ID of the queue item object representing the event in the queue.
 - To process only Archive events, type `-do_archive_events`.
 - To process only Restore events, type `-do_restore_events`.
5. To change the window interval for the Archive tool, use Documentum Administrator.
6. To use the `post_archive`, `pre_restore`, and `post_restore` programs:
 - a. Copy the programs from `%DM_HOME%\bin` (`$DM_HOME/bin`) to `%DOCUMENTUM%\dba` (`$DOCUMENTUM/dba`).
 - b. Edit the programs, if needed.
[Appendix C, Archiving scripts](#), contains the text of these programs. For a description of their purpose and use, refer to [Moving dump files on and off line](#), page 286.

Starting the Archive tool

The Archive tool is active by default. For more information, refer to [Activation](#), page 465.

Restoring documents

When users attempt to open an archived document in a client application, the application prompts them to indicate whether they want to restore the document. If they answer affirmatively, the client application queues a `DM_RESTORE` event. When the Archive tool polls the operator's inbox and finds the event, it will attempt to restore the document.

For a successful restoration, the dump file containing the archived document must be present in the archive directory. You can use a `pre_restore` script to copy the required dump file back to the archive directory if you have moved it to offline storage. Refer to [Moving dump files on and off line](#), page 286 for information about the `pre_restore` script.

Archiving restored documents

Restoring an archived document does not remove the copy of the document from the archive file. It simply copies the document's content back into the original storage area and marks the content on line by setting the `is_offline` attribute in the content object to `FALSE`. If the document is archived again, the Archive tool does not actually dump the document again. It resets the `is_offline` attribute to `TRUE` and removes the content from the storage area. The `set_file` attribute in the content object still contains the full path to the original archive (dump) file.

Custom restoration

If you are not using a Documentum client as your client interface, you can implement automatic restoration using a custom surrogate get feature. Using surrogate get, you can implement the same restoration functionality found in the Documentum clients.

The surrogate get feature is implemented with a program you define and is supported by two attributes defined for the storage area types. The user-defined program is represented in the repository by a `dm_method` object. The storage area attributes are:

- `get_method`
- `offline_get_method`

The `get_method` attribute contains the name of the method object representing the user-defined program in the repository.

The `offline_get_method` attribute is a Boolean attribute that indicates whether the program actually restores the requested document or only queues a `DM_RESTORE` request. You must set it before you execute the user program. Set the value to `FALSE` if the program restores the document and to `TRUE` if the program queues a `DM_RESTORE` event.

When `get_method` is set for a storage area and a user attempts to access an archived document that was originally stored in that area, the server runs the specified program. If `offline_get_method` is `FALSE`, when the program completes, the server assumes that the document is now available and will attempt to fetch the document. If `offline_get_method` is `TRUE`, the server assumes that the document is not available when the program completes and does not attempt to fetch the document.

Full-Text Indexing

This chapter describes the full-text indexing software components and the full-indexes in a repository, and provides information and instructions for the maintenance of full-text indexes. It contains the following topics:

- [Introducing full-text indexing, page 293](#)
- [Overview of the indexing process, page 297](#)
- [What comprises a full-text indexing installation, page 298](#)
- [Managing the index agent and index server, page 303](#)
- [Querying indexes, page 306](#)
- [Maintenance operations, page 306](#)

Introducing full-text indexing

Full-text indexing enables the rapid searching and retrieval of text strings within content files and attributes. A full-text index is an index on the attributes and content files associated with SysObjects and SysObject subtypes. By default, the attributes of all SysObjects and SysObject subtypes are indexed, regardless of data type. If a SysObject has an associated content file, the content file is also indexed, provided the `a_full_text` attribute of the object is set to TRUE and the format of the content file is indexable. If the `a_full_text` attribute is set to FALSE, the content file is not indexed. Content files in all storage areas are indexed.

If you are using distributed content, all content is copied to the primary content store for indexing. The drive on which the primary content store resides must have sufficient space for the primary content store plus the content copied from remote stores for indexing.

During Content Server installation, you are asked to designate those languages for which *grammatical normalization* is enabled. Grammatical normalization ensures that all forms of a word are indexed and that a search for one form of a word also returns other forms of a word. [Choosing parts of speech to index, page 296](#), and the *Content Server*

Full-Text Indexing Installation Guide contain additional information about grammatical normalization.

Full-text indexing is enabled in the repository by default when the repository is created or upgraded to this Content Server version. However, Content Server itself does not create or maintain the full-text index. You must install the full-text indexing software components, which create and maintain the index. For instructions on installing the software and creating the index, refer to the *Content Server Full-Text Indexing Installation Guide*.

Phonetic searching is not supported.

How format objects determine which renditions are indexed

Attributes of the format object determine which formats are indexable and which content files in indexable formats are indexed. If the value of the `can_index` attribute of a content file's format object is set to `TRUE`, the content file is indexable. If the primary content of an object is not in an indexable format, you can ensure that the content file is indexed by creating a rendition in an indexable format.

The `format_class` attribute of the format object may be set to values that determine which formats are indexed:

- `ftalways`

All renditions in formats whose `format_class` attribute is set to `ftalways` are indexed. For example, if a document has renditions in Microsoft Word and PDF formats and the `format_class` attribute for both formats is set to `ftalways`, both renditions are indexed.

- `ftpREFERRED`

If a document has multiple renditions in indexable formats and one is in a format whose `format_class` attribute is set to `ftpREFERRED`, the rendition in that format is indexed rather than any renditions in other formats, with the exception that any formats whose `format_class` attribute is set to `ftalways` are also indexed. If a document has more than one rendition whose `format_class` attribute is set to `ftpREFERRED`, the first rendition processed for indexing is indexed and the other renditions are not. It is recommended that for any document, only one rendition is in a format whose `format_class` attribute is set to `ftpREFERRED`.

If a document has renditions in four different formats, of which the `format_class` of one is set to `ftpREFERRED` and the `format_class` of the other three is set to `ftalways`, all four renditions are indexed.

By default, the first content file in a format whose `can_index` attribute is set to `true` is indexed. Other renditions of the object are not indexed. If the primary content of an object is not in an indexable format, create a rendition in an indexable format. [Appendix F, Supported and Unsupported Formats for Full-Text Indexing](#), contains a complete list of indexable formats.

If the content file associated with a `SysObject` exists in a non-indexable format, its attributes are still indexed. To index the content, create a rendition of the `SysObject` in an indexable format. Use Documentum Content Transformation Services or third-party client applications to create the rendition.

How content file size determines what is indexed

If a content file is larger than 200 MB, only its attributes are indexed. If you are indexing very large files, refer to [Indexing Large Files, page 310](#) for important configuration recommendations.

Which languages are indexed

Content files and attributes in all supported languages are indexed by default. All standard Unicode character sets are supported. No special configuration is necessary. [Appendix G, Supported Languages for Full-Text Indexing](#), contains a complete list of supported languages.

Two right-to-left languages are supported for full-text indexing with certain limitations: Hebrew and Arabic. Other right-to-left languages cannot be indexed. For non-binary formats, only logical text representation is supported; visual text representation is not supported. For those binary formats listed in [Appendix F, Supported and Unsupported Formats for Full-Text Indexing](#), that support right-to-left text in the native format, support is provided for indexing Hebrew and Arabic text, with the exception that PDF files cannot be indexed.

How particular characters are handled

The following Unicode characters are indexed and are searchable:

- Alphabetic characters
- Numeric characters
- Diacritical characters

Diacritical characters linguistically modify the meaning of the characters to which they apply.

- Extender characters

Extender characters extend the value or shape of a preceding alphabetic character. These are typically length and iteration marks.

- Custom characters enclosing Chinese, Japanese, and Korean letters and months

These are derived from a number of custom character ranges that have bidirectional properties, falling in the 3200-32FF range. The specific character ranges are:

- 3200-3243
- 3260-327B
- 327F-32B0
- 32C0-32CB
- 32D0-32FE

Other characters, including punctuation marks and characters such as | and #, are not indexed or searched. Such nonsearchable characters are removed from the indexed text and treated as if they are blank spaces. If a special character is included in a query, it is removed. For example, querying on `Richard+Dodd` would return a document containing the text `Richard=Dodd` because the + and = signs are both replaced by a blank space.

Choosing parts of speech to index

During index server installation, you can designate the parts of speech to be normalized. For example, a search for “cars” also returns “car” if nouns are normalized.

The following combinations of parts of speech can be normalized:

- Nouns
- Nouns and adjectives
- Nouns, adjectives, and verbs
- Nouns and verbs

Grammatical normalization is enabled automatically for Chinese, Japanese, and Korean. Grammatical normalization can be optionally enabled for the following languages:

- German
- English
- Spanish
- French

- Hungarian
- Italian
- Norwegian
- Polish
- Portuguese
- Russian

Overview of the indexing process

Full-text indexing is controlled by three software components:

- Content Server manages the objects in a repository, generates the events that trigger full-text indexing operations, queries the full-text indexes, and returns query results to client applications.

The Content Server uses a query plug-in to send full-text queries to the index server.

- The index agent processes index queue items generated by Content Server and prepares SysObjects for indexing.
- The index server creates full-text indexes and responds to full-text queries from Content Server.

The *Content Server Installation Guide* contains information on installing Content Server. the *Content Server Full-Text Indexing Installation Guide* contains information on installing the index agent and index server.

The indexing process is:

1. A Save, Saveasnew, Checkin, Destroy, Branch, or Prune operation is performed on a SysObject in the repository.
2. The event generates a queue item.
3. The queue item is sent to the full-text user's work queue.

The full-text user, `dm_fulltext_index_user`, is a Superuser created when a repository is created or when an existing repository is upgraded to version 5.3.

4. The index agent acquires the queue item.
5. The index agent retrieves the object associated with the queue item from the repository and, using its index plug-in, creates a *DFTXML* representation of the object that can be understood by the index server.

DFTXML is an XML format that contains the object's attributes and the location of the object's content file, if any.

6. The DFTXML representation of the object is sent to the index server.

7. The index server retrieves the content file, if any, and creates its own representation of the content file and attributes, called FIXML.
8. The index server indexes the content file and the attributes and notifies the index agent that the indexing process is complete.
9. The index agent destroys the queue item for the object.

The full-text index is updated on a continuous basis, provided that all of the software components are running. No special administrative tasks must be performed to ensure that the index is updated and current.

What comprises a full-text indexing installation

A full-text indexing installation consists of the software components (Content Server, the index agent, and the index server), the full-text index, the repository objects required to support the software and the indexing process, and initialization files and initialization file parameters that support full-text indexing.

Content Server

The Content Server generates queue items that are read by the index agent, issues queries to the index server, and returns query results to client applications. The Content Server uses a query plug-in to translate native Documentum full-text queries to the index server's query language.

Index agent

The index agent is a multithreaded Java application running in the Apache Tomcat servlet container. It is installed on the Content Server host or a separate host.

An index agent may run in one of several operational modes:

- *Normal* mode is for event-driven indexing operations.

An index agent associated with a new repository is automatically created in normal mode. In normal mode, the index agent process index queue items and prepares the SysObjects associated with the queue items for indexing.
- *Migration* mode is for creating indexes for a 5.2.5 repository prior to upgrading the repository to 5.3 SP1 or for creating indexes for existing content in an upgraded 5.3 SP1 repository.

In migration mode, the index agent processes all SysObjects in a repository sequentially in `r_object_id` order and prepares them for indexing. A single queue item is used to mark the index agent's progress in the repository.

For complete information on running the index agent in migration mode and creating new indexes, refer to the *Content Server Full-Text Indexing Installation Guide*.

- *File* mode is used to submit a list of objects IDs to the index agent when a new index is created and index verification determines which objects are missing from the index.

Because file mode is used following a migration, its usage is described in the *Content Server Full-Text Indexing Installation Guide*.

An index agent in normal mode is represented by an ft index agent config object. The attributes of the ft index agent config object primarily record status information about the index agent, including the mode in which the index agent is running and when the index agent began processing queue items. The attributes also record configuration information about the index agent, such as the number of queue items processed in a single batch, the number of exporter threads, and the time interval at which the index agent polls the repository for queue items. This information may be viewed using Documentum Administrator. For more information about the ft index agent config object, refer to the *EMC Documentum Object Reference Manual*.

An index agent in migration mode is represented by an XML configuration file, `indexagent.xml`, on the index agent host. Do not modify the parameters in the configuration file unless you are enabling file store mapping. (Mapping file stores for improved indexing performance is documented in the *Content Server Full-Text Indexing Installation Guide*.)

[Index agent modes, page 305](#), contains complete information on the modes in which the index agent runs.

How the index agent processes queue items

When running in normal mode, the index agent processes index queue items and the SysObjects associated with the queue items. The index queue consists of `dmi_queue_items` that are queued to the `dm_fulltext_index_user`. The queue items are generated by Save, Saveasnew, Checkin, Destroy, or Branch events performed on a SysObject or SysObject subtype in the repository. The value of the queue item's `task_state` attribute changes depending on where the SysObject is in the indexing process:

1. The initial `task_state` value is a blank, indicating that the associated SysObject must be indexed and the queue item is available for processing by the index agent.
2. When the index agent acquires the queue item and begins to process the SysObject, the `task_state` is updated to Acquired.

3. When the SysObject is successfully indexed by the index server, the task_state is updated to Done and the queue item is deleted from the repository.

No further updates are made to the queue item.

4. If the index agent encounters a problem in processing the SysObject to which the queue item refers, the task_state is changed to Warning.

This indicates that the index server has made a partial update to the index. The index agent does not further update the queue item or perform further operations on the SysObject. The queue item remains in the queue.

5. If indexing fails and the index server is unable to add information about the SysObject to the index, the task_state is updated to Failed.

The index agent does no further processing on the SysObject and the queue item remains in the queue. To determine which objects failed indexing, use Documentum Administrator.

By default, the index agent acquires queue items in batches of 1,000. If the index agent shuts down unexpectedly after acquiring queue items, the queue items remain acquired by that index agent. When the index agent is restarted, it resets the queue items and processes them correctly.

If the index agent acquires multiple queue items for a particular SysObject, it processes only one of the queue items. For example, if an object is saved several times, the most recent version is indexed. If an object is saved and then deleted, only the delete is processed.

[Managing the index queue, page 316](#), contains complete information on managing the index queue.

Index server and full-text index

The index server is a third-party server product that creates and maintains the full-text index for a repository. The index server also receives full-text queries from Content Server and responds to those queries.

The index server may be installed on the Content Server host or on a different host. For performance reasons, it is strongly recommended that the index server is installed on a different host. The index server is installed to \$DOCUMENTUM/fulltext (%DOCUMENTUM%\fulltext on Windows). For complete information on installing and running the index server, refer to the *Content Server Full-Text Indexing Installation Guide*.

The index server is represented in the repository by the ft engine config object (dm_fulltext_engine_config). There is one instance of the ft engine config object for each fulltext index object.

The index server receives DFTXML from the index agent and creates its own representation of the SysObject to be indexed, called FIXML. If the SysObject has an associated content file, the index server retrieves the content file from the repository. The FIXML is then used to add the SysObject's content and attributes to the index. If the index agent indicates that an object was deleted from the repository, the index server deletes the object from the index.

The index server logs may be viewed from the JSP interface during migration or from Documentum Administrator when the index server is running against a 5.3 or later repository and the index agent is in normal mode.

The full-text index

The full-text index is stored on the index server host, in the \$DOCUMENTUM/data/fulltext (%DOCUMENTUM\data\fulltext on UNIX) , in two directories:

- fixml, which contains the raw data from which the index is created
- index, which contains the full-text index in a group of subdirectories

Repository objects and attributes supporting full-text indexing

Full-text indexing is supported in the repository by objects representing the full-text index, the index server, and the index agent. Attributes of the server config object, SysObject, and queue item also support full-text indexing.

Fulltext index object

A fulltext index object (dm_fulltext_index) represents an index associated with a repository. Its attributes provide information such as the code page in which the Content Server represents strings sent to the query plug-in, the index name, the object ID of the index server for the index, and the name of the location object containing the location of the query plugin. A fulltext index object is created when the first normal mode index agent is configured for the repository. The only attribute of the fulltext index object that may be updated is the ft_engine_id attribute, which contains the object ID of the ft engine config object representing the index server.

FT index agent config object

The ft index agent config object represents a normal-mode index agent configured for the repository. Its attributes record status information and configuration information about the index agent.

FT engine config object

The ft engine config object represents the index server for the repository.

Location objects

Each fulltext index object points to a location object that represents the location of the dmfulltext.ini file.

Supporting attributes of other objects

The following attributes of other objects support full-text indexing:

- a_full_text
- fulltext_location

[How format objects determine which renditions are indexed, page 294](#) contains information on how the setting of the format_class attribute of the format object controls which renditions are indexed.

The a_full_text attribute

The a_full_text attribute is defined for the SysObject type and is inherited by all SysObject subtypes. It is a Boolean attribute that controls whether an object's associated content files are indexed.

When a_full_text is TRUE, content files are indexed whenever a Save, Saveasnew, Checkin, Destroy, Branch, or Prune operation is performed on the object. Any changes to the object's content are added to the index.

The a_full_text attribute is set to TRUE whenever a SysObject is created. Users with Sysadmin or Superuser privileges can change the a_full_text setting. Users without Sysadmin or Superuser privileges cannot change the a_full_text setting.

The `fulltext_location` attribute

The value of the `fulltext_location` attribute of the server config object contains the name of the location object that identifies the fulltext configuration file, `dmfulltext.ini`.

Entries in the `server.ini` file

Two parameters in the `server.ini` file govern aspects of full-text indexing:

- `max_ftacl_cache_size`

Content Server caches ACL information on objects to evaluate security on the results returned by full-text queries. The `max_ftacl_cache_size` key defines the number of elements cached.

The default value is -1 (no limit set). If the value is set to 0, no security information is cached. The value may be set to any integer greater than -1. It is strongly recommended that you do not change the default value.

- `start_index_agents`

The `start_index_agents` parameter defines whether the index agents associated with a particular repository are started at Content Server startup. The default value is `TRUE`. Changing the value to `FALSE` means that no index agent is available to process index queue items in the repository until you manually start an index agent.

The `dmfulltext.ini` file

The `dmfulltext.ini` file is created when the server is installed. It contains information used by Content Server to find the index agent plugin.

Managing the index agent and index server

This section describes starting and stopping the index agent, indicating whether the index agent is started automatically at server startup, and index agent operational modes.

Administering the index agent and index server

Two user interfaces are used for index agent administration:

- In migration mode, a JSP interface is available for starting and stopping the index agent and index server and monitoring the progress of indexing.

For full instructions on using the JSP interface, refer to the *Content Server Full-Text Indexing Installation Guide*.

- In normal mode, use Documentum Administrator for starting and stopping the index agent and index server.

Refer to Documentum Administrator online help for information on starting and stopping the index agent and index server.

Starting and stopping the index agent and index server

The index agent is a Java application running in the Apache Tomcat servlet container. The index server runs in the Apache Web server. Two different interfaces are available for starting and stopping the index agent and index server.

When the index agent and index server are installed, a JSP interface is provided that can be used to start and stop the components. You must use this interface when running the index agent and index server against a 5.2.5 repository.

The interface is accessed at the following URL

```
http://hostname:portno/IndexAgentN/login.jsp
```

where *hostname* is the host where the index agent is installed (use `localhost` if the browser is on the index agent host), *portno* is the port where the index agent listens, which is provided during installation, and *N* is the number of the index agent. Refer to the *Content Server Full-Text Indexing Installation Guide* for full information on using the JSP interface.

When an index agent is running in normal mode against a 5.3 repository, the JSP interface continues to be available, but you can also use Documentum Administrator to start and stop the index agent and index server. Refer to Documentum Administrator online help for instructions.

Stop the index agent before you stop its associated index server. If you try to stop the index server first, Documentum Administrator automatically stops the index agent as well. The JSP interface warns you to stop the index agent before you stop the index server.

Regardless of which interface you use to start and stop the index agent and index server, the Tomcat and Apache processes are still running after you shut down the index agent or index server. You must stop Tomcat and Apache manually. However, if Tomcat is stopped, it must be restarted before the index agent can be started.

The index agent at server startup

By default, at server startup, Content Server checks whether the index agent associated with the repository is started. If the index agent is already running, it remains running. If the index agent is not running and the `start_index_agents` parameter in the `server.ini` file is set to `TRUE` (the default value), the server starts the index agent. If the Tomcat instance in which the index agent runs is not running, Content Server cannot start the index agent.

If the Content Server is shut down while the index agent is running, the index agent shuts down after several attempts to communicate with the server. When the server is restarted, the index agent is started.

Index agent modes

The index agent runs in either migration mode or normal mode against a 5.3 repository and in migration mode against a 5.2.5 repository. Both modes are compatible with file mode, which is used for submitting a list of object IDs to the index agent for indexing.

Normal mode

Content Servers 5.3 and later generate a queue item any time an event such as a checkin requires that a new or modified object must be indexed. In normal mode, the index agent reads the queue item, prepares the object for indexing, and updates the queue item. When the index agent submits the object to the index server for indexing, the index agent deletes the queue item from the repository. The index agent can run in normal mode only against a 5.3 repository.

Migration mode

In migration mode, the index agent prepares all indexable objects in a repository for indexing in object ID order. A single queue item records the ID of the most recent object indexed. The index agent reads the value in the queue item, exports the next object, and updates the queue item. The index agent can run in migration mode to create new indexes against a 5.2.5.x or 5.3 repository.

For example, if you need to move the indexing component installation to a new host computer, you can install the components on the new computer and create a new index there in migration mode, while the existing indexing component installation maintains the existing index in normal mode.

For complete instructions on creating indexes and running the index agent in migration mode, refer to the *Content Server Full-Text Indexing Installation Guide*.

File mode

File mode is used for submitting a list of object IDs to the index agent for indexing.

File mode can be used when the index agent is in any operational mode. However, it is most useful immediately after a new index is created. After a new index is created, the `ftintegrity` tool is run to determine whether all indexable objects in the repository are included in the index. The `ftintegrity` tool produces a list of object IDs of SysObjects that are not included in the index. The list is used to submit the objects to the index agent for indexing. For complete instructions on running the `ftintegrity` tool and running the index agent in file mode, refer to the *Content Server Full-Text Indexing Installation Guide*.

Querying indexes

Indexes are queried using a subset of DQL called FTDQL. Refer to the *DQL Reference Manual* for complete information on FTDQL and on the syntax for full-text queries.

All searching is case-insensitive.

Maintenance operations

This section describes basic maintenance operations for full-text indexing, including the following topics:

- [Turning indexing on and off, page 307](#)
- [Index partitioning, page 307](#)
- [Indexing content in nonindexable formats, page 308](#)
- [Reindexing a repository, page 309](#)
- [Indexing Large Files, page 310](#)
- [Enabling thesaurus searching, page 311](#)
- [Managing the index queue, page 316](#)
- [Pointing a repository to a previously-created index, page 316](#)
- [Obtaining a list of indexable formats, page 317](#)
- [Tracing full-text query operations, page 317](#)
- [Enabling tracing for the index agent, page 317](#)

Turning indexing on and off

The following guidelines apply to turning indexing on and off:

- You can turn off indexing of both attributes and content files.
[Turning off all indexing, page 307](#), contains instructions for turning off indexing.
- You can turn off content indexing but leave attributes indexed.
[Turning off content indexing, page 307](#), contains instructions for turning off content indexing.
- You cannot turn off only attribute indexing.
- You can disable the creation of the full-text index by shutting down the index agent and index server for a repository, but queue items are still generated by the events that trigger indexing.

Turning off all indexing

To turn off all full-text indexing operations in a repository, connect as a Superuser, then unregister the Prune, Save, Saveasnew, Checkin, Destroy, and Branch notifications for the full-text user (`dm_fulltext_index_user`). This disables the generation of queue items for Prune, Save, Saveasnew, Checkin, Destroy, and Branch events.

Turning off content indexing

To turn off content indexing, set the `a_full_text` attribute of the SysObject to FALSE.

Index partitioning

By default, all objects in a repository are indexed into a single partition. A full-text index may optionally be segmented into partitions so that index entries are directed to a particular partition based on the value of the `a_storage_type` attribute of the SysObject.

A partition map defines the index partition to which entries for a storage area's objects are directed. Each partition corresponds to a physical or logical subdivision of an index called a collection. Whether the collections are physical or logical depends on how the index server is configured and your business needs.

Physical partitioning is recommended only if you need to separately back up and restore individual partitions or if you have a very large repository (in this release, 20 million or more documents).

If you are using a consolidated deployment, in which multiple repositories are indexed with a single index server deployment, by default, the deployment uses a logical index server partition and Documentum Professional Services is not required. If you use logical partitioning based on `a_storage_type`, it is recommended that you use physical index server partitioning, which requires Professional Services.

If you require physical partitioning without a consolidated deployment, Documentum Professional Services is required.

Any arbitrary number of storage areas may be mapped to a particular partition. You may map storage areas to partitions on a one-to-one basis, or you may map more than one storage area to a partition. The partition map may be modified after installation to add more partitions. The index agent must be restarted after the partition map is modified.

Index partitioning requires manual configuration of the index server installation before an index is created. Documentum Professional Services is required for implementing index partitioning.

Indexing content in nonindexable formats

Content Server indexes the attributes of all documents with content that have `a_full_text` set to `TRUE` regardless of whether their content files are an indexable format. Content files are indexed if `a_full_text` is set to `TRUE`, the file's format object has `can_index` set to `TRUE`, and the file format is an indexable format. To index documents with contents in non-indexable formats, you must provide an indexable rendition of the content and set three attributes in the primary content's format object. The attributes are `can_index`, `topic_transform`, and `topic_format_name`.

The `can_index` attribute for the primary format must be set to `TRUE` to allow the server to index the renditions. The `topic_transform` attribute is a Boolean attribute. Set this to `TRUE`. The `topic_format_name` attribute defines the indexable format of the rendition. Setting these attributes directs the server to look for a rendition in the format identified in `topic_format_name` and to index that rendition.

For example, the maker format (FrameMaker) is not indexable. Suppose you have documents in the maker format whose content you want to index.

To index those files, you must:

1. Create renditions in mif format of the primary content files for the documents.
2. Add the renditions to the documents.

3. Set `can_index` to `TRUE` for the maker format object.
4. Set `topic_transform` to `TRUE` for the maker format object.
5. Set `topic_format_name` to `mif` format for the maker format object.

If you change the attributes in a non-indexable format to allow indexing of a rendition in an alternate format, the change affects objects in the non-indexable format created after the format object is changed. For example, if you changed the FrameMaker format object (maker) to allow indexing of mif renditions, only Framemaker documents created after the format object was modified are affected. If you want to index the existing FrameMaker documents, you must manually update the `index_format` and `index_formats` attributes in the content objects for those documents. The attributes must be set to the indexable format of the rendition.

By default, indexable renditions created by transforming non-indexable content are stored in the same storage area as the original content. However, you can store them in a separate location. To do so, use Documentum Administrator to set the `rend_backing_store` attribute in the server config object to the chosen location. (For instructions on modifying the server config object, refer to the Documentum Administrator online help.)

Note: Setting `rend_backing_store` does not affect the storage of indexed renditions for content stored in blob or turbo storage areas. These renditions are stored in a directory specified by the `turbo_backing_store` attribute in the server config object.

If you are storing the renditions in a storage area defined by the `rend_backing_store` attribute and the renditions are renditions of encrypted content, be sure to make that storage area an encrypted storage area also. If you do not, the renditions are not encrypted.

Reindexing a repository

An existing index may become corrupted because of disk failure or disk, host failure, or the unexpected shutdown of the index server. When the index is corrupted, it may be desirable to reindex the repository.

You can reindex a repository using the Create Full-Text Events tool (`dm_FTCreateEvents`) in full-reindex mode. The tool is described in the online help system for Documentum Administrator, from which it is run, and in [Create Full-Text Events, page 482](#).

Reindexing the repository rebuilds the existing index by replacing entries in the index.

Indexing Large Files

If you are indexing large files, it is recommended that you modify timeout values in certain index agent and index server configuration files.

To modify index agent and index server timeout values:

1. Connect to the computer where the indexing software is installed as the installation owner.
2. Navigate to `$DOCUMENTUM_SHARED/IndexAgents/IndexAgent1/webapps/IndexAgent1/WEB-INF/classes/` (UNIX and Linux) or `%DOCUMENTUM_SHARED%\IndexAgents\IndexAgent1\webapps\IndexAgent1\WEB-INF\classes\`.
3. Open the `indexagent.xml` file in a text editor.
4. In the `<indexagent_instance>` element, locate the `<runaway_thread_timeout>` and `<runaway_item_timeout>` elements.
5. Replace the default values with `600000000`.
6. In the `<exporter>` element, locate the `<runaway_timeout>` element.
7. Replace the default value with `600000000`.
8. In the `<indexer>` element, locate the `<runaway_timeout>` element.
9. Replace the default value with `600000000`.
10. In the `<fast_indexer>` element, locate the `<fds_callback_wait_time>` element.
11. Replace the default value with `600000000`.
12. Save the `indexagent.xml` file.
13. Stop the index server.
Use the index agent admin tool.
14. Navigate to `$DOCUMENTUM/fulltext/IndexServer/etc/processors/` (UNIX or Linux) or `%DOCUMENTUM%\fulltext\IndexServer\etc\processors\`.
15. Open the `Format.xml` file in a text editor.
16. Locate the `<load module="processorts.Format" class="StellentConverter"/>` element.
17. Increase the timeout value to 3000:

```
<param name="Timeout" value="3000" type="int"/>
```
18. Save the `Format.xml` file.
19. Navigate to `$DOCUMENTUM/fulltext/IndexServer/etc/` (UNIX or Linux) or `%DOCUMENTUM%\fulltext\IndexServer\etc\`.

20. Open the NodeConf.xml file in a text editor.
21. In the Content Distributor section, ensure that the value of the `--operation-timeout` parameter is `6000`.
22. In the Config Server section, ensure that the last entry in the `<parameters>` element is `-p 12000`:


```
<parameters>-P $PORT -s no -M -ORBEndPointNoListen
giop:tcp:pleearth:$PORT1
-ORBEndPointNoPublish giop:tcp::$PORT1 -p 1200<parameters>
```
23. Save the NodeConf.xml file.
24. Restart the index server.

Enabling thesaurus searching

Thesaurus searching allows users to define which words are returned by a particular search.

A synonym file contains the definitions. Typically, entries in the file contain words with similar meanings (synonyms). For example, you can define “auto” and “automobile” as synonym entries that must be returned when the word “car” is the search term. However, an entry can contain unrelated words or create user-defined associations.

A synonym file may also containing spelling variations and abbreviations. For example, “favour” is the British spelling of “favor” and “fvr” is an abbreviation of “favor.”

Thesaurus searching is supported for only one language per index server installation. All languages listed in [Appendix G, Supported Languages for Full-Text Indexing](#), are supported for thesaurus searching. The default language for thesaurus searching is English.

This section contains instructions for creating a synonym file and importing its contents to create a synonym dictionary. Once the synonym file is imported and the synonym dictionary is created, thesaurus searching is enabled. User searches return not only the searched-for word, but also any synonyms defined in the synonym file.

If you upgraded from an EMC Documentum version prior to 5.3, an existing Verity thesaurus file can also be used to create the synonym dictionary.

You can enable thesaurus searching at any time. You can also add entries to an existing synonym dictionary.

Creating the synonym file

The synonym file defines which words you want to be returned by a particular search. Create a text file with synonym entries in the following format:

```
term=[[spelling variations],[abbreviations],[[synonyms][ ]]]
```

The file may have any name. Save the file in the UTF-8 code page.

The brackets in each entry are literal characters that separate spelling variations, abbreviations, and synonyms. The empty brackets at the end of the synonym entry are required.

The following examples contain only synonyms::

```
car=[[ ],[ ],[[automobiles,autos],[ ]]]
cars=[[ ],[ ],[[automobiles,autos],[ ]]]
auto=[[ ],[ ],[[automobiles,autos],[ ]]]
autos=[[ ],[ ],[[automobiles,autos],[ ]]]
```

The following example contains a spelling variation, an abbreviation, and synonyms:

```
favor=[[favour],[fvr],[[prefer,privilege],[ ]]]
```

There are no limitations on the length of an entry or the number of entries in the synonym file.

Importing the synonym file

You can import a synonym file created as described above in [Creating the synonym file, page 312](#) or you can import an existing Verity thesaurus file.

To import the synonym file, connect to the index server host as the index server installation owner and run the ImportDictionary.py script. The script is executed by Cobra, which is included in the index server installation. The following conditions must be met to run ImportDictionary.py:

- The FASTSEARCH environment variable must be set (\$FASTSEARCH on UNIX, %FASTSEARCH% on Windows).
- The script must be run from the \$FASTSEARCH/bin directory (UNIX) or %FASTSEARCH%\bin folder (Windows).
- The synonym file must be found in the \$FASTSEARCH/bin directory (UNIX) or %FASTSEARCH%\bin folder (Windows).

The syntax is

```
cobra ImportDictionary.py [-I filename -M verity|fast -L language -T transaction_type]
```

or

```
cobra ImportDictionary.py [--importfile=filename --mode=verity|fast
--language=language --transactiontype=transaction_type]
```

The arguments are described in [Table 8-1, page 313](#).

Table 8-1. Syntax of ImportDictionary.py script

Argument	Description
-I <i>filename</i> or --importfile <i>filename</i>	The name of the synonym file. May be any arbitrary name.
-M <i>verity fast</i> or --mode <i>verity fast</i>	Whether an existing Verity thesaurus file or a new index server synonym file is being imported. Allowable values are <i>verity</i> and <i>fast</i> . If the argument is not included, the default value is <i>fast</i> .
-L or --language	The language of the synonym file. Allowable values are the names of the languages listed in Appendix G, Supported Languages for Full-Text Indexing . If the argument is not included, the default value is English.
-T <i>transaction_type</i> or --transactiontype <i>transaction_type</i>	The allowable values are: <ul style="list-style-type: none"> • <i>createnew</i>, which instructs the index server to create a new synonym dictionary using the terms in the file <i>filename</i>. • <i>insertentries</i>, which instructs the index server to add entries from the synonym file <i>filename</i> to an existing synonym dictionary. • <i>deletedictionary</i>, which instructs the index server to delete the existing dictionary. Do not provide a file name if you run <code>ImportDictionary.py</code> to delete an existing synonym dictionary. If you do not provide an argument, the English dictionary is deleted by default.

For example, to import a new file called `MySynonyms.txt`:

```
cobra ImportDictionary.py -I MySynonyms.txt -M fast -L English -T createnew
```

To use an existing Verity thesaurus file called `VeritySynonyms`:

```
cobra ImportDictionary.py -I VeritySynonyms -M verity -L English -T createnew
```

To add new synonyms from a file `MoreSynonyms.txt` to an existing synonym dictionary:

```
cobra ImportDictionary.py -I MoreSynonyms.txt -M fast -L English -T insertentries
```

To delete an existing synonym dictionary:

```
cobra ImportDictionary.py -T deletedictionary
```

Logging

Running the ImportDictionary.py script generates a log file in \$FASTSEARCH/bin (UNIX) or %FASTSEARCH%\bin (Windows). The log file is called error.log. It is generated each time the ImportDictionary.py script is run, whether or not errors are reported. The information in the log file is also displayed on your console or monitor while the script is running. The log file contains information on:

- Which options were used to run the script
- The name of the synonym file
- Whether required files are present in the index server installation
- The actions performed by the script
- When any errors occurred
- How to correct the errors
- The name of the synonym dictionary that is created
- Which index server processes create the synonym dictionary

Log sample

The following is a sample log:

```
Information : Logging started
Services ['storageservice', 'j2ee'] will be started
IMPORTANT!!!:The current File C:\Documentum\fulltext\IndexServer\etc\NodeConf.xml
is being modified as C:\Documentum\fulltext\IndexServer\etc\NodeConf.xml.origRestore
it back before rerunning the script if script fails Launching File Transformer
C:\Documentum\fulltext\IndexServer\bin\cobra with args ['convert.py',
'veritysyn.txt', 'FASTFile', 'en']

#####

Information : Launching application C:\Documentum\fulltext\IndexServer\bin\cobra
The argument list is ['cobra', 'convert.py', 'veritysyn.txt', 'FASTFile', 'en']
Successfully generated the transformed file FASTFile
Copying NodeConf.xml.mod as C:\Documentum\fulltext\IndexServer\etc\NodeConf.xml
to start services

#####

Information : Launching application C:\Documentum\fulltext\IndexServer\bin\nctrl
The argument list is ['nctrl', 'reloadcfg']

#####

Information : Launching application C:\Documentum\fulltext\IndexServer\bin\nctrl
```

```
The argument list is ['nctrl', 'start', 'storageservice', 'j2ee']

#####

Information : Launching application C:\Documentum\fulltext\IndexServer\bin\nctrl
The argument list is ['nctrl', 'sysstatus']

#####

Information : Launching application C:\Documentum\fulltext\IndexServer\bin\nctrl
The argument list is ['nctrl', 'sysstatus']
The dictionary app is C:\Documentum\fulltext\IndexServer\bin\dictman
The documentum_synonym_en dictionary will be generated
The DictmanCmdFile command file will be used by dictman
The Dictionary documentum_synonym_en exists
The importFile FASTFile

#####

Starting Dictionary documentum_synonym_en import
Creating Dictman command file DictmanCmdFile

#####

Information : File DictmanCmdFile created
Writing strings
  open /documentum_synonym_en
thesimport /documentum_synonym_en FASTFile
close /documentum_synonym_en
compile /documentum_synonym_en
quit

Generated DictmanCmdFile command file for Dictman

#####

Information : Launching application C:\Documentum\fulltext\IndexServer\bin\dictman
The argument list is ['dictman', '-u', 'root', '-f', 'DictmanCmdFile']
imported entries successfully into Dictionary documentum_synonym_en
Stopping ['qrserver', 'search-1'] services

#####

Information : Launching application C:\Documentum\fulltext\IndexServer\bin\nctrl
The argument list is ['nctrl', 'start', 'qrserver', 'search-1']

#####

Information : Launching application C:\Documentum\fulltext\IndexServer\bin\nctrl
The argument list is ['nctrl', 'sysstatus']
```

```
#####  
Information : Launching application C:\Documentum\fulltext\IndexServer\bin\nctrl  
The argument list is ['nctrl', 'stop', 'qrserver', 'search-1']  
  
#####  
Information : Launching application C:\Documentum\fulltext\IndexServer\bin\nctrl  
The argument list is ['nctrl', 'stop', 'qrserver', 'search-1']  
Error : Application C:\Documentum\fulltext\IndexServer\bin\nctrl returned  
with error code -5  
Error : Launch failed for application C:\Documentum\fulltext\IndexServer\bin\nctrl  
Error : Application C:\Documentum\fulltext\IndexServer\bin\nctrl returned  
with error code -5
```

Managing the index queue

Documentum Administrator provides the interface for viewing the current index queue and managing the items in the index queue.

From the **Indexing Management**→**Index Queue** page, you can:

- View all items in the queue
If you view all items, they can also be sorted by task status.
- View items in the queue by status
- Resubmit all failed queue items or a single failed queue item for indexing
This operation runs the MARK_FOR_RETRY administration method.
MARK_FOR_RETRY updates all queue items with the task_state of failed to NULL, so that the queue items are again acquired by the index agent.
- Delete items of a particular status from the queue

Refer to Documentum Administrator for complete instructions on managing the indexing queue.

Pointing a repository to a previously-created index

If an index is created with the index agent in normal mode, the repository, index agent, and index server are automatically associated with one another.

If an index is created with the index agent in migration mode, a normal mode index agent must be created to update the index. If the repository is version 5.2.5.x, upgrade the repository and then configure a normal mode index agent. If the repository is version

5.3, configure the normal mode index agent. Configuring the normal mode index agent associates the indexing software components, the index, and the repository.

Obtaining a list of indexable formats

Use the following DQL query to obtain a list of content formats that are marked as indexable:

```
SELECT "name", "description" FROM "dm_format"  
WHERE "can_index"=true
```

Tracing full-text query operations

Use the MODIFY_TRACE administration method to turn on tracing for full-text querying operations. The method is executed from Documentum Administrator. Two tracing levels are available: None, in which tracing is turned off, and All, in which all Content Server and full-text messages resulting from queries are logged. The `fttrace_repository.log` file contains messages generated by turning tracing on using MODIFY_TRACE.

When tracing is turned on, FTDQL queries are logged with the following format:

```
FTDQL Execution. ft_convertible = T|F, converted_ft_query  
= %s, ft_search_type = %s
```

When `ft_convertible` is TRUE, `converted_ft_query` contains the converted full-text equivalent of the WHERE clause. If a where clause was not specified in the query's select statement, `converted_ft_query` is empty. The `ft_search_type` indicates how the query was executed against the index.

When tracing is turned on and non-FTDQL queries are executed, the SQL statements created for the DQL queries are logged to the full-text log file.

Enabling tracing for the index agent

Use these instructions to enable tracing for the index agent.

To enable tracing for the index agent:

1. On the index agent host, navigate to the `$DOCUMENTUM_SHARED/config` directory (`%DOCUMENTUM%\config` on Windows).
2. Open the `IndexAgentN.log4j.properties` file, where *N* is the number of the index agent for which you are enabling tracing.

3. Locate the following text:

```
log4j.category.com.documentum=INFO
```

4. Replace INFO with DEBUG:

```
log4j.category.com.documentum=DEBUG
```

5. Save the file.

Users and Groups

This chapter contains information and procedures for administering local users and groups in your repositories. The chapter includes the following topics:

- [User names, page 320](#)
- [User privilege levels, page 321](#)
- [Privileged groups, page 323](#)
- [Adding users, page 324](#)
- [Granting and revoking user privileges, page 331](#)
- [Modifying users, page 333](#)
- [Renaming users, page 334](#)
- [Deleting users, page 334](#)
- [Deactivating, locking, and reactivating users, page 336](#)
- [Adding groups, page 337](#)
- [Modifying groups, page 340](#)
- [Deleting groups, page 341](#)
- [Querying groups, page 343](#)

Use the procedures in this chapter to create and manage local users and groups. To create and manage global users or groups (users and groups common to the repositories in a federation or across all repositories in the enterprise), use:

- The procedures in the *Documentum Distributed Configuration Guide*, or
- The procedures in the LDAP directory server documentation if you have an LDAP directory server installed and the repositories are set up to use it.

User names

User objects represent Documentum users in the Docbase. User objects have four attributes that define user names: `user_name`, `user_os_name`, `user_login_name`, and `user_db_name`. Each attribute has a different purpose.

The `user_name` attribute holds the user's Documentum user name. If you are using Documentum Administrator to create users, this is the name that you assign in the User Name field. The user name can be the same as the `user_os_name` or it can be a different name. If it is different, then it can contain characters such as spaces or apostrophes. For example, Henry VIII or Molly O'Leary are valid Documentum user names. Documentum assigns the `user_name` value to the `owner_name` attribute for new objects created by that user. If your user name is Henry VIII, any objects you create will be owned by Henry VIII. Every user must have a Documentum user name.

The `user_os_name` attribute is the user's name on the operating system, if the user has an operating system account. If the user is an LDAP user or is authenticated with an in-line password, the `user_os_name` attribute may be blank.

The `user_login_name` attribute contains user name that Content Server uses to authenticate the user when he or she connects to a repository. This name is passed to the authentication mechanism (along with the user's password) for authentication whenever the user starts a Documentum session. Every Documentum user must have a `user_login_name`. The default value of the `user_login_name` is the `user_os_name`.

The `user_db_name` attribute holds the user's RDBMS login name. This name is optional for most users. It is only required if:

- The user is a repository owner.
The server uses the repository owner's `user_db_name` attribute value to make connections to the underlying RDBMS.
- The user wants to register tables in the repository.
A user can register any table that the user owns in the underlying RDBMS. To perform this operation, the user must have a `user_db_name`.

It is possible for all four attributes to have the same value. However, if you have assigned a name with spaces or apostrophes, such as Henry VIII or Jane Doe to `user_name`, you cannot assign that name to `user_os_name`, `user_login_name`, or `user_db_name`.

The names in all four attribute must be compatible with the code page defined in the `server_os_codepage` attribute of the server config object. This is the code page of the operating system of the server host machine. If the repository is part of a multi-repository environment and the repositories have different values for the `server_os_codepage`, the names must be ASCII characters only.

User privilege levels

The user privileges define what administrative operations a user can perform in a repository. This section describes the user privileges that may be assigned to users. [Granting and revoking user privileges, page 331](#), contains information on setting user privileges.

Basic user privileges

There are six basic levels of user privileges. The basic privileges define the operations that users can perform on SysObjects in the repository. [Table 9-1, page 321](#), lists the basic user privileges.

Table 9-1. Basic user privileges

Privilege	Description	Corresponding Integer Value
None	User has no special privileges	0
Create Type	User can create object types	1
Create Cabinet	User can create cabinets	2
Create Group	User can create groups	4
Sysadmin	User can <ul style="list-style-type: none"> • Create, alter, and drop users and groups • Create, modify, and delete system-level ACLs • Grant and revoke Create Type, Create Cabinet, and Create Group privileges • Create types, cabinets, and printers • Manipulate workflows or work items, regardless of ownership • Manage any object's lifecycle 	8

Privilege	Description	Corresponding Integer Value
Superuser	<ul style="list-style-type: none"> • Set the a_full_text attribute User can <ul style="list-style-type: none"> • Perform all the functions of a user with Sysadmin privileges • Unlock objects in the repository • Modify or drop another user's user-defined object type • Create subtypes that have no supertype • Register and unregister another user's tables • Select from any underlying RDBMS table regardless of whether it is registered or not • Modify or remove another user's groups or private ACLs • Create, modify, or remove system ACLs • Grant and revoke Superuser and Sysadmin privileges • Grant and revoke extended privileges • View audit trail entries 	16

The None privilege allows a user to perform only the actions allowed by the permissions defined at the object level. Typically, the majority of repository users have the None privilege.

The Create Type, Create Group, and Create Cabinet privileges allow a user to create the item identified by the privilege name. These privileges do not override object-level permissions and having one of the privileges does not automatically bestow any of the others. For example, a user may have the privilege to create cabinets but not object types. Another user may have only the privilege to create groups. A user who creates an object type, group, or cabinet is the owner of the item and can also modify or remove it.

The Sysadmin privilege does not override object-level permissions.

The Superuser privilege gives a user a minimum of Read access to all SysObjects if the ACL does not explicitly grant the user a higher permission. A superuser also always has the Change Permit extended user permission. ([Evaluating a Superuser's permissions, page 395](#), describes how Content Server evaluates the entries in an ACL for a superuser.)

Note: On UNIX platforms, superuser privilege is not the equivalent of root user.

Extended user privileges

There are four extended user privileges. These privileges define security-related operations that users can perform. [Table 9-2, page 323](#), lists the extended user privileges.

Table 9-2. Extended user privileges

Privilege	Description	Corresponding Integer Value
Config Audit	User can execute the Audit and Unaudit methods to start and stop auditing.	8
Purge Audit	User can remove audit trail entries from the repository.	16
View Audit	User can view audit trail entries.	32

These privileges are not hierarchical. For example, granting a user Purge Audit privilege does not confer Config Audit privilege also.

Repository owners, superusers, and users with the View Audit permission can view all audit trail entries. Other users in a repository can view only those audit trail entries that record information about objects other than ACLs, groups, and users.

Only repository owners and Superusers may grant and revoke extended user privileges, but they may not grant or revoke these privileges for themselves.

Privileged groups

Installing Content Server installs some groups whose members are allowed to perform privileged operations even though the members do not have the privileges as individuals. These privileged groups have no default members when they are created. You must populate the groups. [Table 9-3, page 324](#), describes these groups.

Table 9-3. Privileged groups

Group	Description
dm_browse_all	Members of this group can browse any object in the repository.
dm_browse_all_dynamic	This is a dynamic group whose members can browse any object in the repository.
dm_retention_managers	Members of this group can own retainer objects (representing retention policies) and add and remove a retainer from any SysObject. This is a dynamic group.
dm_retention_users	Members of this group can add retainers (retention policies) to SysObjects. This is a dynamic group.
dm_superuserusers	Members of this group are treated as superusers in the repository.
dm_superuserusers_dynamic	A dynamic group whose members are treated as superusers in the repository.

Adding users

Adding a new user to a repository creates a dm_user object in the repository. To add a user to a repository, you must be the repository owner or installation owner, or have Sysadmin or Superuser privileges.

You can use Documentum Administrator, DQL, or the API to add a single user to a repository. Documentum Administrator is the fastest and most convenient way to add one user. If you are adding a large number of users, you can use an LDIF file imported using Documentum Administrator.

New users may also require an operating system user account. If user authentication is performed by Content Server against the operating system, then an operating system account is required. If you use an LDAP directory server for user authentication, an actual operating system account is not required for the user. If the user is authenticated using a password stored in the repository, an operating system account is not required.

A repository includes a group named admingroup which includes all Superusers within the repository. If you grant Superuser privileges to a user after creating or updating the administrative tools, add that user to the admingroup group.

Note: If the repository belongs to a federation and you add a local user with the same name as a global user, the local user is overwritten by the global user when the federation update jobs execute.

Setting user attributes

When you create a new user, you must define the user's:

- User name
- Authentication name
- Email address

The user's name is defined in the `user_name` attribute. The name can be the same as the user's authentication name or it can be a more readable name, such as John Doe. The user name must be unique within the set of user names and group names in the repository.

The authentication name is defined in the `user_login_name` attribute. This name is used by Content Server to authenticate a user when he or she connects to the repository.

The user's address is defined in the `user_address` attribute. The value is the user's email address. The server uses the email address to send electronic mail to the user whenever an event occurs for which the user is registered or a method is executed that generates an email message for the user.

The values for `user_name`, `user_login_name`, and `user_address` must consist of characters that are compatible with Content Server's host operating system code page (the `server_os_codepage` value). If the repository is in a multi-repository environment and the repositories have different `server_os_codepage` values, the values must be ASCII characters only.

In addition, you can also define a variety of other attributes for the user, including :

- Default folder

The default folder is the cabinet or folder in which Content Server stores objects created by the user if the user doesn't identify an alternate storage location. If you don't define a default folder for a user, it defaults to the Temp cabinet.

- Whether a user's access is restricted to particular folders in the repository
- User privileges
- A default ACL
- Client capability
- A home repository

For a complete list of the attributes for the user object type, refer to [User, page 488](#), in the *EMC Documentum Object Reference Manual*. The following sections describe some of the values you can set.

User privileges

A user's user privileges define the operations that a user can perform in the repository. [User privilege levels, page 321](#), lists the valid user privileges.

If you are setting the `user_privileges` attribute directly when you create the user, define the user privileges as an integer value.

To give multiple user privileges by integer value, add the values. For example, to give a user Create Type and Create Cabinet privileges, use the integer 3, the sum of the integers representing the Create Type and Create Cabinet privileges.

You can assign user privileges after you create a user. [Granting and revoking user privileges, page 331](#), contains information on how to do this.

Defining the default ACL

You can set two security attributes for new users: `acl_name` and `acl_domain`. If you do not set `acl_name`, the server creates an ACL to assign to the user and sets `acl_name` and `acl_domain` to the appropriate values for that ACL. That ACL grants the following permissions: `dm_world` receives Read permission; the owner receives Delete permission; and the user's group receives Version permission.

Table 9-4. Security attributes for new users

Attribute	Description
<code>acl_name</code>	The name of an ACL to assign to the user. The ACL can be any ACL you own or any ACL owned by the system (the repository owner). If you do not set this attribute, the server creates an internal ACL in the user's domain as the default ACL.
<code>acl_domain</code>	The name of the user who owns the ACL identified by <code>acl_name</code> . If you do not set this attribute, the server sets it to the repository owner or the user's name, based on the ACL specified in <code>acl_name</code> .

Setting user_db_name

If the user will be a repository owner or wants to register RDBMS tables, you must also set the `user_db_name` attribute. This attribute holds the user's RDBMS login name.

Setting default_folder

The `default_folder` attribute defines the user's default storage folder. A user's default folder is the default storage place for any object the user creates. Depending on how you have set up your site, you may need to create a folder for the user. For example, you may choose to give each user a personal default folder. In this case, it is likely that you will need to create the folder. If several users share a folder, the folder may already exist.

You must specify a folder path identifying a cabinet or folder as the default folder. A folder path has the following syntax:

```
/cabinet_name{/folder_name}
```

You can specify a folder path that does not exist.

Setting accessible folders

A user's repository access can be restricted to designated folders. Set the repeating attribute `restricted_folder_ids` to the full repository paths of each of the folders or cabinets to which the user has access, or use Documentum Administrator.

Setting client capability

The client capability setting is used by Documentum client products, such as Desktop, to determine which functionality to deliver to the user. Content Server does not recognize or use the client capability setting. For information about the client features available with each setting, refer to the Documentum client documentation.

Creating a new user with DQL

To add a user using DQL:

1. To create a default cabinet for the user (if necessary), use the `CREATE...OBJECT` statement.

For example:

```
1>create dm_cabinet object
2>set object_name = 'florabelle'
3>go
```

2. Use the CREATE...OBJECT statement to create the user object and set its attributes.

For example:

```
1>create dm_user object
2>set user_name = 'florabelle',
3>set default_folder = '/florabelle',
4>set user_privileges = 3
5>go
```

The only required attributes are `user_name`, `user_login_name`, and `user_address`. [User](#), page 488, in the *EMC Documentum Object Reference Manual*, contains a list of all user attributes. [Setting user attributes](#), page 325, contains a discussion of other commonly set attributes.

To set additional attributes later or to modify an attribute setting, use the UPDATE...OBJECT statement. [Modifying users](#), page 333, contains more information.

Creating a new user with the API

To use API methods to add a user:

1. As a user with Sysadmin or Superuser privileges, connect to the repository to which you want to add the user.
2. Create and save a default folder for the user. For example:

```
API>create,s0,dm_folder
. . .
returned_folder_id
API>set,c,returned_folder_id,object_name
SET>username
. . .
OK
API>save,s0,returned_folder_id
. . .
OK
```

You can assign any name to the folder. However, if you are giving each user a personal folder, naming the folder after the user makes it easier to find and administer the folder.

3. Create the user object and set its attributes. For example:

```
API>create,s0,dm_user
. . .
returned_user_id
API>set,c,returned_user_id,user_name
```

```

SET>florabelle
. . .
OK
API>set, c, returned_user_id, user_login_name
SET>florabelle
. . .
OK
API>set, c, returned_user_id, user_group_name
SET>engr
. . .

```

The only required attributes are `user_name`, `user_login_name`, and `user_address`. [User, page 488](#), in the *EMC Documentum Object Reference Manual* contains a full list of attributes. [Setting user attributes, page 325](#), contains a discussion of other commonly set attributes.

4. Save the new user object.

```
save, c, returned_user_id
```

When you issue a `Create` method, the server returns the object ID of the newly created object. Subsequent references to that object use the alias last or simply the letter `l` to represent the returned ID. The alias refers to the last referenced object in your session.

Adding multiple users in a single operation

To add multiple users in one operation, you can create a file in LDIF format that defines the users and use Documentum Administrator to read the file and create the users.

This section describes how to create the file. For instructions on importing the file using Documentum Administrator, refer to the Documentum Administrator online help.

LDIF file contents

The LDIF file used to import users contains attribute values for the `dm_user` object representing each user, including the privileges, permits, and group assigned to the user.

The values specified in this file override any values specified in Documentum Administrator. The values you specify in Documentum Administrator are only used if corresponding values are not specified in the file. If the values are not specified in either the file or Documentum Administrator, the server uses the default values listed in [Table 9-5, page 330](#).

Table 9-5. Default values for new users

Argument	Default
user_login_name	<i>username</i>
user_address	<i>username</i>
privileges	0 (None)
folder	<i>/username</i>
group	docu
client_capability	1

There are no default values for *acl_domain*, *acl_name*, and *user_login_domain*.

Setting up the file

The file that Documentum Administrator reads must be in LDIF format. The syntax for each user is:

```
object_type:dm_user
user_name:
user_login_name:
user_address:
user_group_name:
user_privileges:
default_folder:
user_db_name:
description:
acl_domain:
acl_name:
user_os_domain:
home_docbase:
user_state:
client_capability:
```

In the file, the line `object_type:dm_user` marks the beginning of the description of the attributes for a user. Anything between this line and the next line that starts with `object_type` describes one user object for the server to create.

The remaining file entries match the attribute names for the `dm_user` object.

Any ACLs that you identify by `acl_domain` and `acl_name` must exist before you run the file to import the users. Additionally, the ACLs must represent system ACLs. They cannot represent private ACLs.

Any groups that you identify by `user_group_name` must exist before you run the file to import the users.

Content Server will create the default folder if it does not already exist.

Extended characters in the file

If the file contains extended characters, such as an umlaut, the file must be saved as an UTF-8 file.

Granting and revoking user privileges

User privileges define what administrative operations a user can perform in a repository. ([User privilege levels, page 321](#), describes the user privileges.) By default, users have no special privileges in a repository. You can give a user higher privileges when you create the user in the repository or you can do so later. To assign privileges to a user, use Documentum Administrator, DQL, or the API.

Granting and revoking user privileges is subject to the following constraints:

- You must have Sysadmin or Superuser privileges to grant the Create Type, Create Cabinet, or Create Group privileges.
- You must have Superuser privileges to grant Superuser or Sysadmin privileges.
- Only repository owners and Superusers can grant and revoke the extended user privileges, but they may not grant the privileges to themselves or revoke them from themselves.

A user's basic user privileges are stored in the `user_privileges` attribute of the user's `dm_user` object. The user's extended user privileges are stored in the `user_xprivileges` attribute.

Each privilege you grant a user is added to any current privileges. For example, if a user has the Create Type privilege and you grant that user the Sysadmin privilege, the Sysadmin privilege does not replace the Create Type privilege but is added to it. If you later revoke the Sysadmin privilege, the user retains the Create Type privilege.

Superuser privileges and the `admingroup` group

When you grant Superuser privileges to a user, you may also need to add that user to the `admingroup` group to enable them to run jobs or administration methods. The `admingroup` group is created at server installation or upgrade, when the administration tool suite is installed. It contains all the superusers in the repository. At server installation, these are the repository owner and installation owner. After an upgrade, it contains all superusers in the repository.

Members of the `admingroup` group have no inherent privileges other than those they have as superusers. However, administrative jobs, methods, and other related objects use an ACL which restricts their use to the `admingroup` group.

When you revoke Superuser privileges from a user, remember to also remove that user from the `admingroup` group.

[Modifying groups, page 340](#), contains instructions on adding and removing users to and from groups.

Granting and revoking privileges using DQL

Use the DQL GRANT statement to assign user privileges and the REVOKE statement to remove them. You must have Sysadmin or Superuser privileges to grant and revoke the basic user privileges. You must be the repository owner to grant and revoke an extended user privilege.

The syntax of the GRANT statement is:

```
GRANT privilege{,privilege} TO user{,user}
```

The syntax of the REVOKE statement is:

```
REVOKE privilege{,privilege} FROM user{,user}
```

To include multiple privileges in either statement, separate the privileges with commas. Similarly, to identify more than one user, separate the user names with commas. For example:

```
GRANT CREATE TYPE,CREATE CABINET TO jane,jeffrey,ashley
```

or

```
REVOKE CREATE TYPE,CREATE CABINET FROM jane,jeffrey,ashley
```

Granting and revoking privileges using the API

To grant or revoke user privileges through the API, use a Set method. To set the user's basic user privileges, reset the `user_privileges` attribute. To set the user's extended user privileges, set the `user_xprivileges` attribute. The `user_privileges` and `user_xprivileges` attributes are integer attributes. To set them, you must specify the integer value of the privilege or privileges you wish to give the user. ([Table 4-1, page 87](#), and [Table 9-2, page 323](#), list the integer values assigned to the basic and extended user privileges.)

The privilege values are additive. For example, to assign a user the privilege to create types, set the `user_privileges` attribute to 1. To assign the user the privileges to create types and create cabinets, set `user_privileges` to 3 (1 for Create Type plus 2 for Create Cabinet).

Setting `user_privileges` or `user_xprivileges` overwrites the current value of the attribute. To add or revoke a user's privileges, check the value of the attribute first and reset the

value accordingly. For example, suppose you want to add the Create Cabinet privilege (value 2) to a user who already has Create Group (value 4) and Create Type (value 1). The current value in the `user_privileges` attribute is 5 (4+1). To add Create Cabinet, set `user_privileges` to 7. Or, suppose you want to remove the Create Type privilege from that user. To do that, set the `user_privileges` attribute to 4 (subtracting the value of Create Type from the current setting and resetting the attribute to the remainder).

To use API methods to change a user's privileges:

1. As a user with appropriate privileges, connect to the repository that contains the user you want to modify.

2. Obtain the object ID of the user's user object.

```
API>retrieve,c,dm_user where user_name = 'Documentum_user_name'
...
user_object_id
```

3. Fetch the user object.

```
API>fetch,c,user_object_id
```

4. Check the user's current user privileges to ensure that you do not overwrite any existing privileges.

For example:

```
API>get,c,user_object_id,user_privileges
...
integer_value
```

5. Set the new value of `user_privileges`.

For example:

```
API>set,c,user_object_id,user_privileges
SET>new_integer_value
...
OK
```

6. Save the user object.

```
API>save,c,user_object_id
```

Modifying users

You can use Documentum Administrator, DQL, or the API to modify a user's definition in the repository. You must have at least Sysadmin or Superuser privileges to modify a user. Refer to [Granting and revoking user privileges, page 331](#), for information about the privileges needed to grant or revoke user privileges.

If you use Documentum Administrator to change a user's default group, the user is automatically added to the default group unless he or she already belongs to that group.

Using DQL

Use the DQL UPDATE...OBJECT statement to modify a user. [Update...Object, page 158](#), in the *Content Server DQL Reference Manual* contains a description of this statement and its use.

As an example, here is an excerpt from an IDQL session that changes florabelle's default group to support:

```
1>update dm_user object
2>set user_group_name = 'support'
3>where user_name = 'florabelle'
4>go
```

Using the API

If you use the API, you must first check out or fetch the user's object from the repository, then use individual methods to modify or set the desired attributes. After making the changes, use a Checkin or Save to save the changes. Use Checkin if you checked out the object. Use Save if you fetched the object. To obtain the user's object ID (for use in the Checkout or Fetch method), use a Retrieve method. For example:

```
API>retrieve,c,dm_user where user_name = 'florabelle'
```

Renaming users

Use Documentum Administrator to rename a user. Renaming a user changes the user's name recorded in all objects in the repository, including ACLs and alias sets.

Documentum Administrator executes the User Rename administration tool to rename the user. You cannot execute this tool manually. You must use Documentum Administrator to run User Rename. For information about the tool arguments and the generated report, refer to [User Rename, page 534](#).

You cannot rename the repository owner, installation owner, or yourself.

Deleting users

You can delete users using Documentum Administrator, DQL, or the API. However, it is better to deactivate users rather than deleting them. Deactivating a user means that the user is no longer allowed to log into the repository, but the user's user object is not

deleted from the repository. ([Deactivating, locking, and reactivating users, page 336](#), contains more information on this.) You must have Sysadmin or Superuser privileges to delete a user.

When you delete a user, the server does not remove the user's name from objects in the repository such as groups and ACLs. Consequently, when you delete a user, you must also remove or change all references to that user in objects in the repository. For example, every SysObject or SysObject subtype in the repository has an attribute called owner_name that identifies the user who owns that object. Before you delete a user from a repository, update the objects created by the user, changing the owner_name to identify a different user.

You can delete a user and then create a user with the same name. If you add a new user with the same name as a deleted user and have not removed references to the deleted user, the new user inherits the group membership and object permissions belonging to the deleted user.

If you delete a user, the server also removes all registry objects that reference the user as the subject of audit or event notification requests.

You cannot delete the repository owner, installation owner, or yourself.

Using DQL

Use the DQL DELETE...OBJECT statement to delete a user. The qualification in the statement's WHERE clause identifies the user you want to delete. For example, here is an excerpt from an IDQL session that deletes the user florabelle:

```
1>delete dm_user object
2>where user_name = 'florabelle'
4>go
```

Refer to [Delete...Object, page 89](#), in the *Content Server DQL Reference Manual* for the full syntax.

Using the API

If you use the API, use a Destroy method to delete a user. You must know the object ID of the user's user object. To obtain the object ID, use a Retrieve method. For example:

```
API>retrieve,c,dm_user where user_name = 'florabelle'
```

Deactivating, locking, and reactivating users

By default, users are created in the repository in the active state. Active users may be deactivated or locked and, if needed reactivated. A user's activation state is recorded in the `user_state` attribute of the user's user object. Active users may log into the repository and perform those operations for which they have permissions. Deactivated or locked users cannot log into the repository.

It is more convenient to deactivate or lock a user than to delete a user because it allows you to stop a user's access to the repository without requiring you to also modify all the documents or other objects that might reference the user in an attribute.

Deactivating or locking users

You must have Sysadmin or Superuser privileges to deactivate or lock a repository user.

To deactivate or lock a user, use Documentum Administrator or DQL to change the user's login state. In Documentum Administrator, you can choose either Inactive, Locked, or Inactive and Locked to deactivate a user.

If you are using an LDAP directory server to manage users, deleting the user's LDAP entry may automatically deactivate the user the next time the `dm_LDAPsynchronization` job runs. (This depends on the particular directory server you are using and how your site is configured. [How the directory server and a repository are synchronized](#), page 356, contains more details.)

A user may be deactivated automatically if the value in the user's `failed_auth_attempt` attribute exceeds the maximum number of failed authentication attempts defined in the repository configuration, in the `max_auth_attempt` attribute of the `docbase` config object. ([Limiting authentication attempts](#), page 375, describes how this feature works.)

If you deactivate or lock a user who is currently logged in, the user is allowed to complete the repository session but can only perform Assume or Disconnect operations. After disconnecting, the user is not allowed to log in subsequently.

If necessary, a user with Superuser privileges can use a ticket to log in as a deactivated or locked user, and a user with Superuser privileges can add a deactivated or locked user to a group.

You cannot change the login state of the repository owner, installation owner, or yourself.

Reactivating users

Use Documentum Administrator or DQL to reactivate a user that is in the inactive or locked state. You must have Sysadmin or Superuser privileges to reset the user to active.

If the user was inactivated automatically because he or she experienced more than the maximum number of failed authentication attempts when logging in, reactivating the user resets the value in the user's failed_auth_attempt attribute to 0.

Using DQL to change a user's login state

Use the DQL UPDATE...OBJECT statement to change a user's login state. To deactivate the user, set the user_state attribute to 1, 2, or 3. Setting the attribute to 1 sets the user state to Inactive. Setting it to 2 sets the user state to Locked. And, setting it to 3 sets the user's state to Inactive and Locked.

For example, to set the user to Inactive, the syntax is:

```
UPDATE "dm_user" OBJECT
set "user_state" = 1
WHERE "user_name"='user name'
```

To activate a user, set the attribute to 0:

```
UPDATE "dm_user" OBJECT
set "user_state" = 0
WHERE "user_name"='user name'
```

For example, here is an excerpt of an IDQL session that deactivates the user florabelle:

```
1>update dm_user object
2>set user_state = 1
3>where user_name = 'florabelle'
4>go
```

For a complete description of the statement, refer to [Update...Object](#), page 158, of the *Content Server DQL Reference Manual*.

Adding groups

You can use Documentum Administrator, DQL, or the API to create a group. The easiest way to add a group is using Documentum Administrator. To create a group, you must have Create Group, Sysadmin, or Superuser user privileges.

Note: If the repository belongs to a federation and you add a local group with the same name as a global group, the local group is overwritten by the global group when the federation update jobs execute.

By default, a group is owned by the user who creates it. To assign group ownership to another individual user, you must be a superuser. To assign group ownership to a group, you must be a member of the group to which you are assigning ownership.

Groups can be either public or private. By default, groups created by a user with Create Group privilege are private, and groups created by a user with Sysadmin or Superuser privileges are public. To change the default, set the `is_private` attribute to the appropriate value. If `is_private` is TRUE, the group is a private group. If `is_private` is FALSE, the group is a public group.

If the Collaborative Services with Rooms feature is enabled, groups can be assigned to a Room and designated *room private groups*. A room private group is any group for which the `group_native_room_id` attribute has a value. A room private group must also have a `group_display_name` assigned that is unique within the Room, and the `group_display_name` together with `group_native_room_id` must be unique in the repository. (For more information on Collaborative Services, refer to the documentation for Documentum Webtop.)

A group can also be dynamic. A dynamic group is a group whose member list is a list of potential members. By default, a new group is not a dynamic group. ([Dynamic groups, page 379](#), contains more information about dynamic groups.) If you define the group as a dynamic group, then you have the option of defining whether members are considered members by default or non-members by default.

The default membership setting for dynamic groups controls whether group members are considered in or out of the group when a session is started. It is FALSE by default, meaning that users are not considered members by default, and an application must issue a session call to “add” the user to the group at connection time. (This setting is recorded in the `is_dynamic_default` attribute.) If the members are considered in the group by default, then the application must issue a session call to “remove” a user from the group if needed. [Changing the membership setting of a dynamic group, page 342](#), contains instructions on changing the default membership setting.)

The members of a group can be individual users or other groups. A non-dynamic group cannot contain a dynamic group as a member. A dynamic group can contain other dynamic groups or non-dynamic groups as members. (If a non-dynamic group is a member of a dynamic group, the members of the non-dynamic group are treated as potential members of the dynamic group.)

The name you assign to a group must consist of characters compatible with the Content Server’s server OS code page.

Content Server makes no use of the `is_private` flag. The flag is provided so that user-written applications can determine which groups to display under circumstances that the applications define.

Using DQL

Use DQL to create a group if you want to accept the default group ownership and group type. (A group is owned by its creator by default and is created as a standard group by default.) If you want to create a group owned by another user or by a group or want to create a group whose class is role or domain, use Documentum Administrator to create the group.

Use the DQL CREATE...GROUP statement to create a group. The syntax is:

```
CREATE [PUBLIC|PRIVATE] GROUP group_name
[WITH] [ADDRESS email_address] [MEMBERS members]
```

members is a comma-separated list of the users and groups that belong to the group. Identify users by their *user_name* values. Identify groups by the *group_name* values. For example, the following statement creates a group called editors and adds johnr, sallyt, and haroldw to the group:

```
CREATE GROUP "editors" ADDRESS "editors" MEMBERS johnr,sallyt,haroldw
```

Alternatively, you can use a SELECT statement to obtain the member names. For example, assuming that *user_dept* is a user-defined attribute for the *dm_user* type:

```
CREATE GROUP "editors" ADDRESS "editors"
MEMBERS (SELECT "user_name" FROM "dm_user"
WHERE "user_dept" = 'documentation')
```

Include the PUBLIC keyword to create the group as a public group. Include PRIVATE to create the group as a private group. Refer to [Adding groups, page 337](#), for information about public and private groups.

Using the API

To use API methods to add a group:

1. Connect to the repository as a user with appropriate privileges.
2. Create the group object and set its attributes. For example:

```
API>create,s0,dm_group
. . .
returned group id
API>set,c,returned group id,group_name
SET>editors
. . .
OK
API>set,c,returned group id,group_address
SET>editors
. . .
OK
API>set,c,returned group id,users_names
SET>johnr
```

```
. . .  
OK  
API>append,c,returned group id,users_names  
SET>sallyt  
. . .
```

When you are creating groups, you must set the group's `group_name` attribute. A group's name must be unique among the groups and users in the repository. (Note that Content Server stores group names in lowercase.) Generally, you also set the `users_names` attribute to add members to the group. This is a repeating attribute. If you use Set to add the first member, use Append to add subsequent members. Append automatically adds the members to the end of the list of members in the attribute.

If you do not want to accept the default setting for the `is_private` attribute, set that attribute also. ([Adding groups, page 337](#), contains more information about public and private groups.)

The group address is an optional attribute, but one that must be set if the group has a UNIX user account.

3. Save the new group object.

```
save,c,returned group id
```

When you issue a Create method, the server returns the object ID of the newly created object. Subsequent references to the object can use the alias `last` or simply the letter `l` to represent the returned ID. This alias refers to the last referenced object in your session.

Modifying groups

You may need to modify a group's definition in the repository. For example, you may want to change the email address, add or remove a member, or change the default membership setting. You can use Documentum Administrator, the DQL ALTER GROUP statement, or the API to modify a group.

To modify a group, you must be one of the following:

- The group's owner
- A superuser
- A member of the group that owns the group to be modified
- Identified in the group's `group_admin` attribute, either as an individual or as a member of a group specified in the attribute

Only a superuser can change the ownership of an existing group. Only a superuser or the owner of a group can change the `group_admin` attribute of a group.

To add a deactivated user to a group, you must have Superuser privileges.

Using DQL

Use a DQL ALTER GROUP statement to modify a group. For example, here is an excerpt from an IDQL session that adds florabelle to the engr group and changes the group's mail address:

```
1>alter group engr add florabelle
2>alter group engr set address base_engr
3>go
```

For a complete description of the statement and its use, refer to [Alter Group, page 43](#), of the *Content Server DQL Reference Manual*.

Using the API

If you use the API, you must check out or fetch the group's object ID, then use individual methods to set attributes, and checkin or save the object to the repository. Use a Checkin method if you checked out the group object. Use a Save method if you fetched the group object. You must have the object ID of the group object to execute the methods. To obtain the object ID, use a Retrieve method. For example:

```
API>retrieve,c,dm_group where group_name = 'engr_group'
```

Deleting groups

You can use Documentum Administrator, DQL, or the API to delete groups.

When you delete a group, Content Server does not remove references to the group's name from other objects in the repository. Consequently, before you delete a group, if you do not intend to create another group with the same name, remove all references to the group in repository objects. Groups are referenced by name in a variety of attributes in repository objects. For example, a group may be referenced as the owner of an object, a member of another group, or a value in an alias set.

Deleting a group does remove all registry objects that reference the group as the subject of an audit or event notification request.

To delete a group, you must be one of the following:

- The group's owner
- A superuser
- A member of the group that owns the group to be modified (if the group is owned by a group)



Caution: Do not delete the `admingroup` group from the repository. The `admingroup` group, which includes all superusers within the repository, is used by the jobs that make up the administration tool suite.

Using DQL

Use the DQL `DROP GROUP` statement to delete a group. The syntax is:

```
DROP GROUP group_name
```

group_name identifies the group you want to delete. For example, here is an excerpt from an IDQL session that deletes the group `enr`:

```
1>drop group enr
2>go
```

For a complete description of this statement and its use, refer to [Drop Group, page 93](#), of the *Content Server DQL Reference Manual*.

Using the API

If you use the API, use the `Destroy` method. You must have the object ID of the group object to execute the method. To obtain the object ID, use a `Retrieve` method. For example:

```
API>retrieve,c,dm_group where group_name = 'enr_group'
```

Changing the membership setting of a dynamic group

Dynamic groups allow you to determine whether users in the group's membership list are considered members of the group or not by default when the users connect to the repository. This behavior is controlled by the setting in the `is_dynamic_default` attribute of the group object.

When a dynamic group is created, the attribute is set to `F` by default. That setting means that users are not considered members of the group automatically. When the user connects to the repository, the application he or she is using must issue an `IDfSession.addDynamicGroup` call. If you reset the attribute to `T`, the server considers the users to be group members by default. To remove the user from the group, the application must issue an `IDfSession.removeDynamicGroup` method at session startup.

To reset that attribute, use Documentum Administrator.

Querying groups

This section describes how to obtain two kinds of commonly needed information about groups:

- Names of users in a particular group
- List of the groups in a repository and a count of users in each.

Obtaining a list of members in a group

To obtain a list of users in the group, you can query the `_all_users_names` computed attribute. This attribute returns a list of all users directly or indirectly contained in a group.

The first time you query `_all_users_names` in a session, the attribute's value is cached on the server and client side. If the attribute is queried again in the session, the return values are retrieved from the cache.

If the group membership is changed during the session, the cached values are invalidated and the value is re-computed when you query the attribute again. However, any changes made to the group by users in other, concurrent sessions aren't visible to your current session's server process. Consequently, to ensure that the values returned are correct, you can issue a `Revert` method on the group object before querying the computed attribute. Issuing the `Revert` method invalidates the cached copy, forcing the server to recompute the attribute's value.

Obtaining a list of groups with a count of the members in each

You can use the following query to obtain a list of groups in a repository and a count of the members in each group:

```
SELECT gr2.i_superuser_names, count(gr1.users_names)
FROM dm_group_r gr1, dm_group_r gr2
WHERE gr1.r_object_id=gr2.r_object_id AND
gr2.i_superuser_names IS NOT NULL
```

```
GROUP BY gr2.i_supergroups_names
```

Managing User Authentication

This chapter describes the options available for user authentication and how to implement them. The chapter includes the following topics:

- [Authentication options, page 345](#)
- [Using the default authentication mechanism, page 347](#)
- [Using a custom external password checking program, page 350](#)
- [Using Windows domain authentication for UNIX users, page 351](#)
- [Using an LDAP directory server, page 354](#)
- [Using authentication plug-ins, page 366](#)
- [Using an in-line password, page 370](#)
- [Trusted logins, page 371](#)
- [Unified logins, page 371](#)
- [Managing encrypted passwords, page 371](#)
- [Limiting authentication attempts, page 375](#)

Authentication options

User authentication typically occurs when a user attempts to connect to a repository. (It can also occur when another product, such as Document Control Manager, requires user authentication prior to performing an operation.) Content Server determines whether the user is a valid, active repository user and, if so, authenticates the user name and password. Documentum supports a variety of options for implementing user authentication. You can perform user authentication using

- The default mechanism
- A custom `dm_check_password` program
- An LDAP directory server
- A authentication plug-in

- An in-line password

Default mechanism

The default mechanism authenticates the user against the operating system. This implementation is the simplest to set up. to [Using the default authentication mechanism, page 347](#), contains information about how the default mechanism works and how to set it up.

Custom password checking program

You can create a custom password checking program and set up the servers to call that program for user authentication. This option is particularly useful if you want to use Windows domain authentication for UNIX users. [Using a custom external password checking program, page 350](#), contains instructions on creating custom password checking programs.

LDAP directory server

LDAP is the acronym for Lightweight Directory Access Protocol, a TCP/IP-based protocol for communication between a client (in this case, Content Server) and a directory server. Documentum supports several directory servers (refer to the Content Server Release Notes for a complete list). If you use a directory server, you have the following options:

- Authenticate against the directory server directly, using a secure or a non-secure connection
- Authenticate using an LDAP-enabled `dm_check_password` program

[Using an LDAP directory server, page 354](#), contains complete information about LDAP support and how to set up and use a directory server with a repository.

Authentication plug-in

Authentication plug-ins provide an alternate way to customize user authentication. Documentum provides one authentication plug-in with Content Server. This plug-in allows you to use the Netegrity SiteMinder Policy Server with Content Server. The plug-in supports Web-based Single Sign-On (SSO) and strong authentication. (Strong authentication is the use of authentication tokens such as smart cards or biometrics.)

[Using the netegrity plug-in, page 368](#), contains a listing of which platforms are supported by the plug-in and instructions on using the plug-in.

You can write and install your own authentication plug-in. [Implementing a custom authentication plug-in, page 368](#), contains instructions.

In-line password

A user can be authenticated using an encrypted password that is stored in the `user_password` attribute of the user object. [Using an in-line password, page 370](#), contains instructions for using in-line passwords.

Using the default authentication mechanism

The default authentication mechanisms for UNIX and Windows are different.

UNIX platforms

The default authentication mechanism is an external password checking program. The program must be external because the operating systems on which Content Server is supported use C-2 security level password validation. At this level, user passwords are not stored in an `etc/passwd` file, but in a separate file that require root level privileges to access. These files are:

- `/etc/shadow` on Solaris
- `/.secure/etc/password` on HP-UX
- `/etc/security/passwd` on AIX

Documentum provides a default program called `dm_check_password`. During server installation, the program is copied to the `$DOCUMENTUM/dba` directory. The program creates a location object, named `user_validation_location`, that stores the location of the program and sets the `validate_user` attribute in the server config object to the name of the location object. When a user is authenticated, the server uses the location object named in the `validate_user` attribute to find the password checking program.

Both the source code and `dm_check_password` is provided as both source code and compiled code. If you have a C compiler, you can recompile the program and use the recompiled version.

To use the default mechanism on UNIX, repository users must have operating system accounts.

You can create your own password checking program. [Using a custom external password checking program, page 350](#), contains instructions.

You can also configure Content Server to use the password checking program to authenticate UNIX users against Windows domain. [Using Windows domain authentication for UNIX users, page 351](#), contains instructions.

Windows platforms

On Windows platforms, the default authentication is an internal process. Content Server does not use `dm_check_password`. When Content Server is installed on a Windows platform, the `validate_user` attribute in the server config is not set.

To use the default mechanism on Windows, repository users must have an operating system account.

Authenticating in domains

On Windows platforms, if you are authenticating against the operating system, Content Server authenticates the user's operating system name and password within a domain. A user's domain is stored in the user's `user_os_domain` attribute. A default domain is defined for all users in the repository in the `server.ini` file, in the `user_auth_target` key.

Whether the server uses the user-specific domain or the default domain for authentication depends on whether the server is running in domain-required mode or not.

The Content Server installation procedure installs a repository in no-domain required mode. If your Windows configuration has only one domain whose users access the repository, the no-domain required mode is sufficient. If the users accessing the repository are from varied domains, using domain-required mode provides better security for the repository.

No-domain required mode

In no-domain required mode, users are not required to enter a domain name when they connect to the repository. In this mode, a user's operating system name must be unique among the `user_os_name` values in the repository.

How the server authenticates a user depends on what value is found in the user's `user_os_domain` attribute. The attribute can contain an asterisk (*), a blank, or a domain name:

- If `user_os_domain` contains an asterisk or blank, Content Server authenticates the user using operating system name and the domain specified in the connection request. If no domain is included in the connection request, the server uses the domain defined in the `user_auth_target` key in the `server.ini` file.
- If `user_os_domain` contains a domain name, Content Server authenticates against the domain identified in the `user_os_domain` attribute.

Domain-required mode

In domain-required mode, users must enter a domain name or the name of an LDAP server when they connect to the repository. The domain value is defined when the user is created and is stored in the `user_login_domain` attribute in the `dm_user` object.

In domain-required mode, the combination of a user's login name and domain or LDAP server name must be unique in the repository. This means it is possible to have multiple users with the same user login name if each user is in a different domain. For example, the repository could have mark in the finance domain and mark in the engineering domain. It is also possible for one user to be several users in the repository—mark in finance and mark in engineering could be the same person.

Trusted logins are the only exceptions to the rule about supplying a domain name. Trusted logins do not require a domain name even under domain-required mode.

Determining the repository's authentication mode

If you are unsure which mode the repository is running in, you can examine the `auth_protocol` attribute in the `docbase` config object. If the repository is in the default mode, this attribute is blank. If the repository is in the domain-required mode, this attribute's value is `domain-required`.

Converting to domain-required mode

After a repository is changed to the domain-required mode, the only way to return it to the default mode is to create a new repository and dump and load the domain-required mode into the new repository.

Documentum provides the `dm_domain_conv` script to convert to domain-required mode. You cannot change to domain-required mode by setting any attributes manually. The script not only resets the `auth_protocol` attribute, but also recreates some indexes used by Content Server. The script is found in `%DM_HOME%\install\tools`.

To change to domain-required mode:

1. Log into the repository as a user with Sysadmin or Superuser privileges.
2. Execute the `dm_domain_conv` script using the following command line:

```
dmbasic -f dm_domain_conv.ebs
```

The script prompts you for the repository to convert, the name of a user with Superuser privileges, the Superuser's password, the Superuser's domain, the users' domain, and a log file for output.

Using a custom external password checking program

You can write and install a custom password checking program to use regardless of the platform on which the server is running.

This section outlines the basic procedure for creating and installing a custom password checking program. Documentum provides standard technical support for the password checking program that is created and provided with the Content Server software, as part of the product release. For assistance in creating, implementing, or debugging a custom password checking program, contact Documentum Professional Services or Documentum Developer support.

Basic steps

The following procedure outlines how to implement a custom password checking program. Use Documentum Administrator to perform steps 3 and 4.

To use a custom password checking program:

1. Write and compile the program.
2. Put the program in `%DOCUMENTUM%\dba ($DOCUMENTUM/dba)`.
3. Create a new location object to point to the location of the custom program.
4. Set the `validate_user` attribute in the server config to the name of the new location object.

Writing the program

A custom program must

- Accept the same input arguments as the default `dm_check_password` program.
- Return the same values as the `dm_check_password` program.

Additionally, if the program is to be installed on a UNIX host, the program must meet the following requirements:

- The program must be owned by root
- Its group ownership must be the Documentum system administrator's group (`admingroup`)
- Its permissions must be set to 4550

Setting the permissions to 4550 means that only the program's owner and members of the administrator's group can run the program and that the program will set UID on execution.

Using Windows domain authentication for UNIX users

You can configure Content Server to authenticate UNIX users against a Windows domain. To do so, you must:

1. Create and install a custom `dm_check_password`.
2. Set the `auth_protocol` attribute in the `docbase` config object to `unix_domain_used`.
3. Set up a domain controller map.
4. Modify the `user_source` attribute for the users in the repository.

Note: UNIX users who are authenticated against a Windows domain cannot execute methods under their own accounts. All methods executed by such users must be run with `run_as_server` set to `TRUE`.

Modifying the `dm_check_password` program

You must obtain the source code for SMB from GNU in order to recompile the `dm_check_password` program so that it can authenticate users against a Windows domain.

You can use GNU's C compiler, which is called `gcc`, or you can use the C compiler supplied by your operating system vendor. You must have GNU's `gmake` utility.

Documentum Technical Support provides no support for Steps 1 and 2 in the following procedure.

To modify `dm_check_password`:

1. Obtain the GNU SMB library source code.
This is available at Samba sites on the Web.
2. Build the GNU SMB library.
3. Copy the `smbvalid.a` library from the `/smbval` directory to the `$DM_HOME/install/external_apps/checkpass` directory.
4. Copy the `valid.h` file from the `/include` directory to the `$DM_HOME/install/external_apps/checkpass` directory.
5. In the `$DM_HOME/install/external_apps/checkpass` directory, open the `make_check_prog` script.
6. In the `make_check_prog` script, set the `do_domain` variable to `-Ddomain_authentication`.

```
set do_domain = -Ddomain_authentication
```
7. Set the `domain_lib` variable to `smbvalid.a`.

```
set domain_lib=smbvalid.a
```
8. Follow the other instructions in the comments to the `make_check_prog` script.
9. Run the `make_check_prog` script.
A new `dm_check_password` program is produced.
10. Copy `dm_check_password` to the `$DOCUMENTUM/dba` directory.
11. Log in as root.
12. Change the ownership of `dm_check_password` to root:

```
chown root dm_check_password
```
13. Change `dm_check_password`'s group to the group defined for the Documentum installation.

```
chown group_name dm_check_password
```
14. Change the permissions on `dm_check_password`:

```
chmod 6711 dm_check_password
```

Setting auth_protocol

Use Documentum Administrator to modify the docbase config object to set the `auth_protocol` attribute to `unix_domain_used`. For instructions, refer to the Documentum Administrator online help.

Setting up the domain controller map

A domain controller map provides information about domain controllers to Content Server. In a repository, a domain controller map is stored in a `dm_auth_config` object. `dm_auth_config` objects are created by Documentum Administrator when you use Documentum Administrator to set up the domain controller map. A repository has only one `dm_auth_config` object. The object type has three repeating attributes:

- `domain_name`, which stores the names of Windows domains
- `primary_controller`, which stores the names of the primary controllers for the domain
- `backup_controller`, which stores the names of the backup controllers for the domain

The values at a particular index position in the `primary_controller` and `backup_controller` attributes identify the primary and backup controllers for the domain at the corresponding index position in `domain_name`.

To define a domain controller map, modify the docbase config object using Documentum Administrator. The values entered in all three attributes must be in lowercase. Additionally, do not enter a fully qualified name for a primary or backup domain controller. Enter only the host name. For complete instructions, refer to the Documentum Administrator online help.

Setting the user_source attribute

The value in the `user_source` attribute is used by Content Server to determine which authentication mechanism to apply to the user. To direct Content Server to use a windows domain for a UNIX user, set the `user_source` attribute for the user to one of the following:

You must modify each UNIX user in order to enable Windows domain authentication.

- `domain only`
Indicates that a user is authenticated only against a Windows NT domain.
- `UNIX first`
Indicates that a user is authenticated first against the UNIX password file or NIS database, and if that fails, the user is authenticated against a Windows domain.

- domain first

Indicates that a user is authenticated first against a Windows domain, and if that fails, the user is authenticated against the UNIX password file or NIS database.

Use Documentum Administrator to modify the user's attribute. For instructions, refer to the Documentum Administrator's online help.

Using an LDAP directory server

An LDAP directory server is a third-party product that maintains information about users and groups. (Refer to the *Content Server Release Notes* for a list of the directory servers supported with Content Server.) Documentum Content Servers use LDAP directory servers for two purposes:

- To manage users and groups from a central location
- To authenticate users

It isn't necessary for all users and groups in a repository to be managed through an LDAP directory server. A repository can have local users and groups in addition to the users and groups managed through a directory server.

You can use more than one LDAP directory server for managing users and groups in a particular repository.

This section provides information and instructions about using an LDAP directory server for user authentication.

Benefits

Using an LDAP server provides a single place where you can make additions and changes to users and groups. The changes from the directory server are automatically propagated (using a job) to all the repositories using the directory server.

Additionally, you can map user object attributes to LDAP attributes or constant values. When the user is imported into the repository or updated from the directory server, the mapped attributes are set to the values of the LDAP attributes or the constant. The mappings are defined when you define the LDAP setup values. You can also change them later, adding additional mapped attributes, changing their mapping, or deleting mappings.

Note: Content Server requires three attributes to be defined for a users: `user_name`, `user_login_name`, and `user_address`. When you define an LDAP configuration object in Documentum Administrator, default mapped values are provided for these attributes. You can change the defaults or define values for the attributes for each LDAP user, but

you must provide some value or mapping for these attributes. Users cannot be saved to the repository without values for these three attributes.

Using an LDAP directory server to manage users and groups in the Documentum system ensures that:

- The users and groups defined in the directory server are in each repository using the directory server
- The values of the mapped attributes for the users are the same in each participating repository

Note: Documentum does not support the use of dynamic groups on an LDAP server.

Constraint

The `Changepassword` API method is not supported for users managed through an LDAP directory server.

LDAP authentication options

To authenticate an LDAP user, Content Server can perform the checking itself or invoke an external password checking program called `dm_check_password`. You can use the external password checking program provided by Documentum or create a custom program.

If Content Server performs the checking, you can config the LDAP server to use a secure connection with Content Server. (You cannot use a secure LDAP connection if you are using an external password checking program.)

If a secure connection is in use, when Content Server connects to the directory server for the first time, the directory server identifies itself by returning its certificate to Content Server. Content Server verifies the certificate against a certificate database. This is called SSL server authentication. Documentum implements secure LDAP connections using Netscape/iPlanet CSDK.

[Implementing an LDAP directory server, page 358](#), contains instructions for setting up use of an LDAP directory server.

Directory servers and federations

In a federation, only the governing repository can communicate with an LDAP server for the purposes of managing users and groups. Entries in the directory server are

synchronized with the governing repository and it is the responsibility of the governing repository to propagate the changes to the member repositories.

To support user authentication, the member repositories must contain local LDAP configuration objects that point to the directory server used by the governing repository or a local replicated directory server. (A replicated directory server is a feature of directory servers. It is not replicated using Content Server's object replication.)

How the directory server and a repository are synchronized

Content Server uses the `dm_LDAPSynchronization` job to synchronize the entries in the directory server and the repository. This job is part of the Content Server's administration tool suite. The job is installed in the inactive state. ([Dm_LDAPSynchronization, page 491](#), contains details about the job and [Activating the dm_LDAPSynchronization job, page 362](#), describes how to activate it.)

The `LDAPSynchronization` job updates the repository to match the entries in the directory server. This is a one-way operation. Changes in the directory server are propagated to the repository. Changes in the repository are not propagated to the directory server.

The first time the `LDAPSynchronization` job is run, non-directory users and groups in the repository are matched to directory users and groups in the LDAP directory server, converting them to directory users and groups and updating them from the directory server. First-time synchronization can be used optionally in future synchronizations. [Enabling first-time synchronization rules, page 364](#), contains information on using first-time synchronization.

Depending on the directory server and its configuration, the job can perform the following operations in synchronizations subsequent to the first:

- Import into the repository any new users and groups in the directory server.
- Rename in the repository any users and groups whose names are changed in the directory server.

For users, the renaming occurs if a change is detected in the LDAP attribute to which the user's `user_name` attribute is mapped. For groups, renaming occurs if a change is detected in the `cn` (common name) to which the group's `group_name` attribute is mapped.

- Inactivate users in the repository if they are deleted in the directory server.

If you are using Netscape iPlanet Directory Server, Oracle Intranet Directory Server, or Microsoft Active Directory with a repository on a Windows host, the job can perform all operations. When using iPlanet, you must enable the changelog feature to use the renaming and inactivation operations. (Instructions for enabling the changelog feature

are found in the vendor's iPlanet Administration Guide.) The renaming and inactivation operations are not supported on Microsoft Active Directory for repositories on UNIX hosts.

You can also define the operations performed by setting attributes in the associated LDAP configuration object or by setting command line arguments for the job. For example, you can direct the job to only import and change users or only groups. You can disable the inactivation updates or the renaming changes.

When the job creates a new user in the repository, it also creates the user's default folder and group if needed and sets any user attributes that are mapped to LDAP attributes or a constant. (To create the default folder and group, those repository attributes must be mapped to LDAP attributes that define the folder and group for the user. If no such mapping exists for the folder, the job sets the user's default folder to /Temp.)

When the job creates a new group in the repository, it populates the group with the members defined in the LDAP group. If the users in the group do not exist in the repository, the job creates them in the repository.

For rename operations, the LDAPSynchronization job creates a job request for each name change and marks the job to run immediately. The job is run as soon as the agent exec process picks up the request. The LDAPSynchronization job does not wait for the rename jobs to complete.

Note: If you delete an LDAP user or group, the repository user or group is not deleted. Doing so would cause referential integrity problems in the repository. However, if the inactivation operation is available and in use, deleting an LDAP user will set the repository user inactive.

user_source attribute and the dm_LDAPSynchronization job

The dm_LDAPSynchronization job sets the user_source attribute of users to LDAP when it updates or creates users in a repository. Content Server determines which authentication mechanism to use for a particular user by examining the value of the user_source attribute (unless a plug in module is specified explicitly in the connection request). For LDAP users, user_source must be set to LDAP.

Integrating an LDAP directory server with a repository

When you add an LDAP directory server to an existing Documentum installation, the users and groups defined in the LDAP directory server are given precedence. If a user or group entry in the LDAP directory server matches a user or group in the repository, the

repository information is overwritten by the information in the LDAP directory server when the dm_LDAPsynchronization job executes.

All users in a repository must have three required attributes set. The attributes are user_name, user_login_name, and user_address. Users defined in the LDAP directory server must have LDAP attributes that can be mapped to these attributes. The LDAP configuration set up provides mapped defaults for the required repository attributes or you can define custom mappings. Refer to the Documentum Administrator online help for instructions.

If your LDAP directory server is Active Directory, user_name must be mapped to either a common name (CN) or a display name. If user_name is not mapped to either a CN or display name, the repository inactivation operation does not function properly for the users.

Implementing an LDAP directory server

This procedure creates an LDAP config object and sets the server's ldap_config_id attribute in the server config to the object ID of that object. If you have multiple ldap config objects in your repository, Content Server uses the ldap config object identified in the ldap_config_id attribute of its server config object.

To use an LDAP directory server with a repository:

1. Install the LDAP directory server and add the user and group entries.
Refer to the documentation from the directory server's vendor for instructions on installation and adding entries. Documentum does not support dynamic LDAP groups.
2. Define the LDAP set-up values for the repository.
Use the LDAP configuration facilities in Documentum Administrator to define the LDAP set-up values. Refer to [Defining the set-up values, page 359](#), for information about the set-up values. (Refer to the Documentum Administrator online help for instructions on using the user interface.)
3. To use external password checking:
Note: This option is not supported if you are using a secure connection.
 - a. If the program will run on a UNIX platform, use the procedure in [Building and installing an LDAP-enabled password checking program \(UNIX only\), page 362](#), to build and install an LDAP-enabled dm_check_password program.
 - b. Set the use_ext_auth_prog attribute in the LDAP config object.
 - c. Set user_validation_location in the server config object to the name of the location object pointing to the location of the dm_check_password program

4. To use a secure connection:

Note: This option is not supported if you are using external password checking.

- a. Enable secure connections using the SSL server authentication feature of the LDAP directory server.
Refer to the documentation accompanying the directory server for instruction on enabling secure connections for the server. You must enable the SSL server authentication option.
 - b. Configure the LDAP set-up values for a secure connection.
[The secure connection attributes, page 361](#), contains information about the requirements.
 - c. Download the certutil utility and the issuing Certificate Authorities all the way up to the self-signed root certificate.
[Downloading certutil and the certificate authorities, page 361](#), contains information about obtaining the certutil utility and the Certificate Authorities.
 - d. Install the certificate database and Certificate Authorities.
[Installing the certificate database and CA certificates, page 361](#), contains instructions.
5. Activate the dm_LDAPSynchronization job in the repository after ensuring that the default schedule meets your needs.
The default schedule for the synchronization job is once a day at 4 a.m. [Activating and scheduling administration tools, page 465](#), contains instructions on changing the schedule. [Activating the dm_LDAPSynchronization job, page 362](#), contains instructions on activating the job.

Defining the set-up values

Defining the set-up values creates an LDAP config object. To define these values, you must have Superuser privileges and must use Documentum Administrator.

This section provides information and guidelines for defining the following set-up values:

- [Distinguished name and bind type, page 360](#)
- [Search bases and filters, page 360](#)
- [The secure connection attributes, page 361](#)

Distinguished name and bind type

As part of the definition, you are asked to provide a distinguished name and password for the Content Server. The name and password are used to ensure that the client asking for user authentication is a valid client of the LDAP directory server. The client, either Content Server or the check password program, must provide a distinguished name and password to validate its request.

The distinguished name and password for the client are encrypted and stored in `%DOCUMENTUM%\dba\config\docbase_name\ldap_ldapconfigid.cnt` (`$DOCUMENTUM/dba/config/docbase_name/ldap_ldapconfigid.cnt`). The file system permissions on this file allow only the Documentum installation owner to access the file.

You are also asked what bind type you want to use. The bind type defines how the client obtains a user's distinguished name for authentication purposes. If the `bind_type` attribute is set to `bind_search_dn`, the client uses the user's operating system name to obtain the distinguished name from the directory server. If the `bind_type` attribute is set to `bind_by_dn`, the user's distinguished name is obtained directly from the user's `user_ldap_dn` attribute. The `bind_type` attribute defaults to `bind_search_dn`.

For more information about binding names, refer to the LDAP directory server's documentation.

Search bases and filters

The set up values also include a definition of the search bases and filters. A search base identifies the point in the LDAP schema at which the search for a particular user or group begins. A filter is a defined condition that confines the users or groups in the search to a particular set.

Oracle Intranet Directory uses indexes to make directory attributes available for searches. Some attributes are indexed by default. However, there are three attributes that you must explicitly index if you are using Oracle Intranet Directory as the LDAP directory:

- `createTimestamp`
- `modifyTimestamp`
- `operationTime`

The `createTimestamp` and `modifyTimestamp` attributes must be indexed in order for the LDAP synchronization job to run correctly. The `operationTime` attribute must be indexed if you have chosen the Deactivate users option for the LDAP configuration. In addition to these attributes, you can also index additional attribute for use in a search filter. Refer to the *Oracle Intranet Directory Admin Guide* for instructions on indexing attributes.

For more information about search bases and the format of a filter definition, refer to the LDAP directory server's documentation.

The secure connection attributes

There are three set-up values that must be defined to use a secure LDAP connection:

- SSL mode
- SSL port
- Certificate Database Location

SSL mode, represented in the ldap config object by the `ssl_mode` attribute, defines whether the LDAP server is using a secure or non-secure connection. You must set this when defining the LDAP set-up values. To configure a secure connection, chose Secure as the SSL mode. When you do, the interface lets you edit the SSL port field.

SSL port, represented in the ldap config object by the `ssl_port` attribute, identifies the port the LDAP server uses for the secure connection. This value is 636 by default. You can reset this.

Certificate Database Location, represented in the ldap config object by the `certdb_location` attribute, identifies the location of the certificate database. The attribute value is the name of the location object pointing to the certificate database. The value is `ldapcertdb_loc`. The directory that `ldapcertdb_loc` points to is `%DOCUMENTUM%\dba\secure\ldapdb` (`$DOCUMENTUM/dba/secure/ldapdb`).

The certificate database location is set automatically when you define the LDAP set-up values, which creates the ldap config object.

Downloading certutil and the certificate authorities

The `certutil` utility is used to create a certificate database and load CA certificates into the database. Download the `certutil` utility from the following site:

<ftp://ftp.mozilla.org/pub/mozilla.org/security/nss/releases/>

Copy the utility to `%DM_HOME%\bin` (`$DM_HOME/bin`).

You can obtain information about the utility from:

<http://www.mozilla.org/projects/security/pki/nss/tools/certutil.html>

Download CA certificates from the Web site of the vendor who provided the directory server with the SSL server certificate. You must download the Root Certificate Authority and the issuing certificate authorities all the way up to the self-signed root certificate.

Installing the certificate database and CA certificates

Use the `certutil` utility to install the certificate database and the CA certificates.

To install the certificate database and CA certificates:

1. Create the cert7.db.

On Windows:

```
certutil -N -d %DOCUMENTUM%\dba\secure\ldapdb
```

On UNIX:

```
certutil -N -d $DOCUMENTUM/dba/secure/ldapdb
```

2. Add the Root Certificate Authority to the database and provide the necessary trust level.

```
certutil -A -n "documentum ldap root" -t "C,C,C" -i rootcert.crt
```

3. Install the remaining CA certificates in the chain (if there are any):

```
certutil -A -n "documentum ldap sub root" -t "C,C,C" -i subrootcert.crt
```

Activating the dm_LDAPSynchronization job

The dm_LDAPSynchronization job is installed in the inactive state. To activate the job, use Documentum Administrator. By default, the dm_LDAPSynchronization job executes once a day at 4 a.m when activated. If needed, change the schedule before activating the job. [Activating and scheduling administration tools, page 465](#), contains information on how to change the schedule. Refer to Documentum Administrator online help for instructions on activating a job.

Building and installing an LDAP-enabled password checking program (UNIX only)

Use the following procedure to build and install an LDAP-enabled password checking program. This procedure is only necessary if you want to run an LDAP-enabled password checking program on a UNIX platform.

To build an LDAP-enabled password checking program:

1. Log in to the Documentum installation as the installation owner.
2. Change directory to the \$DM_HOME/install/external_apps/checkpass directory.
3. Open the make_check_prog script in a text editor.
4. Set the do_ldap variable to -Dldap_authentication.

```
do_ldap=-Dldap_authentication
```
5. Set the ldap_lib variable to \$DM_HOME/bin.

```
ldaplibdir=$DM_HOME/bin
```

6. Run the `make_check_prog` script.
7. Copy the `dm_check_password` program to `$DOCUMENTUM/dba`.
8. Log in to the host as the root user.
9. Change the ownership, group, and permissions of the `dm_check_password` program:

```
chown root dm_check_password  
chgrp group_name dm_check_password  
chmod 6750 dm_check_password
```

group_name is the name of the default group.

Note on using Active Directory

If you are using Active Directory to perform LDAP user authentication (`user_source` is set to LDAP), the Active Directory can be configured in either mixed mode or native mode.

If you are not using LDAP authentication but have Active Directory set up to perform domain authentication (`user_source` is not LDAP), Active Directory must be configured in mixed mode.

Synchronizing users between scheduled synchronization jobs

Content Server supports a feature allowing directory users to be synchronized in the repository between scheduled synchronization jobs. The feature is controlled by the setting of the `dir_user_sync_on_demand` attribute of the `docbase` config object.

The LDAP synchronization job runs periodically at intervals you set. If users added to a directory server attempt to connect to the repository before the next time the synchronization job runs, by default the users cannot be authenticated because they are not yet repository users. When the `dir_user_sync_on_demand` attribute is set to T (TRUE) and a user attempt to connect to the repository, the server searches all available LDAP directory servers for the user. (If the user provided a domain at connection time, only the directory server designated is searched.) If the user is found and authenticated, the user record is immediately copied from the directory server to the repository.

Using multiple LDAP directory servers

A repository can be configured to use multiple LDAP directory servers. This feature is controlled by two attributes in the server config object:

- `ldap_config_id`, which identifies the default LDAP config object from which the server synchronizes users and groups
- `extra_directory_config_id`, a repeating attribute that identifies the additional LDAP config objects from which the server synchronizes users and groups

When a new LDAP config object is created using Documentum Administrator, designate the configuration object as the default object or an extra configuration object. The new LDAP config object is associated with the server to which you are connected when you log in to Documentum Administrator.

You can configure the LDAP synchronization job to synchronize with any or all of the directory servers associated with a server config object, using the `source_directory` argument. [Dm_LDAPSynchronization, page 491](#) contains instructions.

Deleting an LDAP directory server from a repository

Deleting an LDAP config object from a repository removes the connection to the LDAP directory server represented by the config object. All users and groups synchronized from that LDAP directory server are either deleted from the repository or converted to non-directory users and groups. When a user is converted to a non-directory user, the `user_source` attribute of the user object is set to inline password. The user is not automatically supplied with a password and cannot connect to the repository until a password is assigned by a System Administrator or Superuser or until the user account is converted back to a directory user.

Enabling first-time synchronization rules

To force the first-time synchronization rules to be used on a subsequent synchronization, set the `first_time_sync` attribute in the ldap config object to T (TRUE). You can do this using DQL or by checking **First time sync** in Documentum Administrator.

Binding LDAP users to a different directory server

You can bind existing LDAP users to a directory server different from their current directory server.

To bind LDAP users to a different directory server:

1. Delete the LDAP config object for the current directory server.
Use Documentum Administrator to delete the LDAP config object. This converts the users and groups to non-directory users.
2. Ensure that the users and groups are represented in the new directory server.
3. Create a new LDAP config object corresponding to the new directory server.
4. Enable the LDAP synchronization job.
When the synchronization job runs, the first-time synchronization rules are used. The non-directory users and groups are matched with the directory users and groups and converted to directory users and groups.

Listing certificates in the certificate database

You can execute the certutil utility with the following command line to obtain a listing of the certificate authorities currently installed in the certificate database:

```
certutil -L -d <cert7.db_directory_path>
```

For example, on Windows:

```
certutil -L -d %DOCUMENTUM%\dba\secure\ldapdb
```

For example, on UNIX:

```
certutil -L -d $DOCUMENTUM/dba/secure/ldapdb
```

Troubleshooting the synchronization job

To troubleshoot the LDAPSynchronization job, you can examine the regular job report and the trace_ldapsync_dfc.txt and trace_ldapsync_dmcl.txt files. The trace files are generated regardless of whether tracing is turned on for the job. These files are stored in the %DOCUMENTUM%\share\temp\ldif\<docbase_name> (\$DOCUMENTUM/share/temp/ldif/<docbase_name>) directory.

The job also creates the regular job trace file if the method_trace_level argument in the job is set to a value greater than zero. Both the report and log file can be easily accessed from Documentum Administrator. [Reports and trace log files, page 462](#), contains instructions on viewing job reports and log files and information about where they are stored in the repository.

Examining the ldap config object can also be useful when troubleshooting. To obtain a dump of the ldap config object, execute the following commands in IAPI:

```
API>retrieve,c,dm_method where object_name='dm_LDAPsynchronization'  
API>dump,c,l
```

Obtaining the Content Server and java version in use can also be helpful. You can obtain the java version using one of the following commands at the operating system prompt:

```
java -fullversion
```

or

```
jview -version
```

Finally, the `ldapsearch` utility, if available with your directory server, is a useful tool. You can pass the search base, class, and filter values from the ldap config object to the utility and generate output that lists all users and groups that the LDAPsynchronization job will import into the repository.

Using authentication plug-ins

Content Server supports the use of authentication plug-ins. Authentication plug-ins are implemented as DLLs or shared libraries, depending on the platform hosting the plug-in.

One plug-in, supporting Netegrity, is provided with Content Server. [Using the netegrity plug-in, page 368](#), describes how to use that plug-in.

You can also write and install custom modules. [Implementing a custom authentication plug-in, page 368](#), describes how to write and install a custom module.

Plug-in scope

You can use a plug-in to authenticate users connecting to a particular repository or to any repository in the installation. The choice is defined by where the modules are installed. All plug-in modules are installed in a base directory. If they are installed directly in the base directory, they are loaded by the servers for all repositories in the installation. If they are installed in repository-specific directories under the base directory, they are loaded only by the servers for the specific repositories.

When you install Content Server, the procedure creates the default base directory, `%DOCUMENTUM%\dba\auth` (`$DOCUMENTUM/dba/auth`). When a repository is created, the procedure creates a subdirectory under the base directory specific to the repository. The repository configuration procedure also creates a location object, called `auth_plugin`, that points to the base directory and sets the `auth_plugin_location` attribute in the server config object to the name of the location object.

Any plug-in installed in `%DOCUMENTUM%\dba\auth` (`$DOCUMENTUM/dba/auth`) is loaded into every server that starts, for all repositories in the installation. To use a

plug-in only with a particular repository, place the plug-in in the repository-specific subdirectory under %DOCUMENTUM%\dba\auth (\$DOCUMENTUM/dba/auth). For example, if you want to use the Netegrity plug-in with a repository called engr_db, move the Netegrity module to the %DOCUMENTUM%\dba\auth\engr_db (\$DOCUMENTUM/dba/auth/engr_db) directory.

When a Content Server starts, it loads the plug-ins found in its repository-specific directory first and then those found in the base directory. If two or more plug-ins loaded by the server have the same identifier, only the first one loaded is recognized. The remaining plug-ins with the same name are not loaded.

Identifying a plug-in for use

There are two ways to identify a plug-in to use for authentication:

- Include the plug-in identifier in the connection request arguments
- Set a user's user_source attribute to the module's plug-in identifier

When Content Server receives a connection request, it checks to determine whether a plug-in identifier is included in the arguments. If not, the server examines the user's user_source attribute to determine which authentication mechanism to use.

To use a plug-in to authenticate users for connection requests issued by an application, the application must prepend the plug-in identifier to the password argument before sending the connection request to the DMCL.

Set the user_source attribute to a plug-in identifier when you want to use a plug-in to authenticate a particular user regularly.

Defining a plug-in identifier

Plug-in identifiers are defined as the return value of the dm_init method in the plug-in module's interface. A plug-in identifier must conform to the following rules:

- It must be no longer than 16 characters.
- It cannot contain spaces or non-alphanumeric characters.
- It cannot use the prefix dm_. (This prefix is reserved for Documentum.)

For example, the following are legal identifiers:

- myauthmodule
- authmodule1
- auth4modul

To include a plug-in identifier in a connection request, the application must prepend the following syntax to the password argument:

```
DM_PLUGIN=plugin_identifier/
```

Plug-in identifiers are accepted in all methods that accept a password as an argument (Assume, Authenticate, Connect, Changepassword, and Signoff).

Using the netegrity plug-in

Documentum provides an authentication plug-in supporting authentication against a Netegrity SiteMinder Policy Server. The plug-in enables validation of netegrity tokens sent to Content Server by Web-based clients.

Note: The Netegrity plug-in is not supported on the HP Itanium and Linux platforms.

Documentum client products, such as Desktop Client and Webtop, do not support the use of the Netegrity Plug-in out of the box. You must customize the client to use this plugin. For assistance in creating, implementing, or debugging the customization, contact Documentum Professional Services or Documentum Developer Support.

The Content Server installation procedure stores the plug-in in the %DM_HOME%\install\external_apps\authplugins (\$DM_HOME/install/external_apps/authplugins) directory. There is a README.txt file that describes how to install the plug-in. [Table 10-1, page 368](#), lists the files for the plug-in on the supported platforms.

Table 10-1. Netegrity plug-in files

Module	File
Netegrity	dm_netegrity_auth.dll (Windows) dm_netegrity_auth.so (Solaris) dm_netegrity_auth.sl (HP-UX on PA-RISC) dm_netegrity_auth.a (AIX)

The directory also includes the Netegrity header files and libraries, to allow you to rebuild the plug-in if you wish.

To use the plug-in after you install it, include the plug-in identifier in connection requests or set the user_source attribute in users to the plug-in identifier. ([Identifying a plug-in for use, page 367](#), contains more information.)

The plug-in identifier for the Netegrity plug-in is dm_netegrity.

Implementing a custom authentication plug-in

You can write and install custom authentication plug-ins. On a Windows platform, the plug-in must be a DLL. On a UNIX platform, the plug-in must be a shared library.

Authentication plug-ins that require root privileges to authenticate users are not supported. (If you want to write a custom authentication mechanism that requires root privileges, use a custom external password checking program. [Using a custom external password checking program, page 350](#), contains instructions.)

This section outlines the basic procedure for creating and installing a custom authentication plug-in. Documentum provides standard technical support for plug-ins that are created and provided with the Content Server software, as part of the product release. For assistance in creating, implementing, or debugging a custom authentication plug-in, contact Documentum Professional Services or Documentum Developer support.

To implement a custom authentication plug-in:

1. Write the plug-in.
[Writing the authentication plug-in, page 369](#), contains instructions.
2. Install the plug-in.
[Plug-in scope, page 366](#), contains information about where to install an authentication plug-in.
3. Enable its use.
[Identifying a plug-in for use, page 367](#), contains information about enabling plug-in use.
4. Restart Content Server to load the new plug-in.

Writing the authentication plug-in

To write an authentication plug-in, you must implement the following interface:

```
dm_init(void *inPropBag, void *outPropBag)
dm_authenticate_user(void *inPropBag, void *outPropBag)
dm_change_password(void *inPropBag, void *outPropBag)
dm_plugin_version(major, minor)
dm_deinit(void *inPropBag, void *outPropBag)
```

The `inPropBag` and `outPropBag` parameters are abstract objects, called property bags, used to pass input and output parameters.

The `dm_init` method is called by Content Server when it starts up. The method must return the plug-in identifier for the module. The plug-in identifier should be unique among the modules loaded by a server. If it is not unique, Content Server uses the first one loaded and logs a warning in the server log file.

`dm_authenticate_user` performs the actual user authentication.

`dm_change_password` changes a user's password.

`dm_plugin_version` identifies the version of the interface in use. 1.0 is the only supported version.

`dm_deinit` is called by Content Server when the server shuts down. It frees up resources allocated by the module.

You can find detailed comments on each of the interface methods in the `dmauthplug.h` header file. All authentication plug-ins must include this header file. It is found in `%DM_HOME%\install\external_apps\authplugins\include\dmauthplug.h` (`$DM_HOME/install/external_apps/authplugins/include/dmauthplug.h`).

Additionally, all plug-ins must link to the `dmauthplug.lib` file. (On Solaris, the `dmauthplug.lib` file is named `dmauthplug.a`.) This file is found in `%DM_HOME%\install\external_apps\authplugins\include` (`$DM_HOME/install/external_apps/authplugins/include`).

Internationalization

An authentication plug-in can use a code page that differs from the Content Server code page. To enable that, the code page must be passed in the output property bag of the `dm_init` method. If the code page is passed, Content Server translates all parameters in the input property bag from UTF-8 to the specified code page before calling the `dm_authenticate_user` or `dm_change_password` methods. The server also translates back any error messages returned by the plug-in. A list of supported code pages is included in the header file, `dmauthplug.h`.

Tracing authentication plug-in operations

Plug-ins are responsible for writing their own trace files. The trace level is determined by the `DM_TRACE_LEVEL` parameter in the input property bag. The initial value of the parameter is taken from the server start up flag `-otrace_authentication`. However, if a user issues a `SET_OPTIONS` administration method that changes the trace authentication level, the new level will be reflected in the plug-in tracing.

The suggested location of the trace file is defined by the `DM_LOGDIR_PATH` parameter in the `dm_init` method.

Using an in-line password

To authenticate a user with an encrypted password stored in the repository, the `user_password` attribute of the user object must be set. Use Documentum Administrator,

the API, or DQL to set the password. When users are created using an imported LDIF file, passwords cannot be set in the LDIF file. You must set the password manually for any user created with an LDIF file.

Trusted logins

Trusted logins occur when the client is running on the same machine as the Content Server, the client user is the installation owner, and the installation owner's domain is the same as that defined in `user_auth_target`.

Unified logins

Unified login is a feature available for users on Windows platforms. It allows the users to connect to a repository through Documentum Desktop using their Windows login credentials.

To enable unified login:

1. Set the `a_silent_login` attribute of the server config object to T.
2. Set the Use Windows Login option on the Desktop Client client interface for each repository for which you want to use unified login.

Refer to the Desktop Client online help for instructions.

Managing encrypted passwords

Content Server and many of the internal jobs that manage repository operations use passwords stored in files in the installation. These passwords are stored in encrypted format by default. The passwords are encrypted using the AEK when you install Content Server or create the job. [Table 10-2, page 372](#), lists the password files whose content is encrypted by default. All the files are found in `%DOCUMENTUM%\dba\config\docbase_name` (`$DOCUMENTUM/dba/config/docbase_name`). You must be the repository owner to access or edit these files.

Table 10-2. Password files encrypted by default

File	Description
dbpasswd.txt	This file contains one line with the database password used by Content Server to connect to the RDBMS. (This is password for the repository owner.)
<i>docbase_name</i> .cnt	The file contains one line with the password used by an object replication job and the distributed operations to connect to the repository as a source or target repository. If this file is present, the <i>dm_operator.cnt</i> file is not.
dm_operator.cnt	The file contains one line with the password used by an object replication job and the distributed operations to connect to repositories. If this file is present, <i>docbase_name.cnt</i> files are not used.
federation.cnt	Contains the information, including passwords, used by a governing repository server to connect to member repositories. The file is stored with the governing repository. The format of the file's content is: <i>member_repository_name:user_name:password:[domain]</i>
ldap_object_id.cnt	Contains the password used by Content Server to bind to an LDAP server.

Using encryptpass

Use Encryptpass to encrypt any password that you want to pass in encrypted form to the one of the following methods:

- Assume
- Authenticate
- Changepassword
- Connect
- Signoff

Passwords encrypted with Encryptpass cannot be decrypted explicitly by an application or user. There is no method provided to perform decryption of passwords encrypted with Encryptpass. The DMCL decrypts those passwords internally when it encounters one in the arguments of one of the above methods.

Passwords encrypted with Encryptpass are prefixed with DM_ENCR_PASS.

For complete information about using this method, refer to [Encryptpass, page 201](#), in the *Content Server API Reference Manual*.

If you do not want to use encrypted passwords

If you do not want to use an encrypted password for a particular operation, use a text editor to edit the appropriate file. (Table 10–2, page 372, contains a list of the files.) Remove the encrypted password and replace it with the clear text password.

Changing an encrypted password

If you find it necessary to change one of the encrypted passwords described in Table 10–2, page 372, use the `dm_encrypt_password` utility to do so. This utility takes an unencrypted password, encrypts it, and writes it to a specified file. If the file is one of the password files maintained by Content Server, the utility replaces the current encrypted password in that file with the new password. You must be the repository owner to use this utility.

To encrypt or change a password in a password file maintained by Content Server, use the following syntax:

```
dm_encrypt_password [-location AEK_location][-passphrase [passphrase]]
-database repository_name -remote remote_repository_name |
-operator | -redbms [-encrypt password]
```

To create or change an encrypted password in a file that you have created, use the following syntax:

```
dm_encrypt_password [-location AEK_location][-passphrase [passphrase]]
-file file_name [-encrypt password]
```

The arguments have the following meanings:

`-location AEK_location` Identifies the location of the AEK file to be used to encrypt the password. If this argument is not set, the environment variable `DM_CRYPTPO_FILE` must be set.

`-passphrase passphrase` Specifies the passphrase used to protect the AEK file.

If the argument is included without a passphrase, the utility prompts for a passphrase.

If the argument is not included, the utility attempts to use the default passphrase. (The default passphrase can be defined when the `dm_crypto_boot` utility is run to set up the AEK.)

If a default passphrase is not defined, the utility checks the shared memory location, based on the location argument or the default

- location, for an AEK or passphrase. If neither is found in shared memory, the utility exits with an error.
- docbase *repository_name* Identifies the repository for which the password is being encrypted.
Do not include this argument if you include the -file argument.
 - remote *remote_repository_name* Identifies the file to operate on as %DOCUMENTUM%\dba\config*repository_name*\remote_*repository_name*
You must include the -docbase argument if you include -remote.
 - operator Identifies the file to operate on as %DOCUMENTUM%\dba\config*repository_name*\dm_operator.cnt
You must include the -docbase argument if you include -operator.
 - rdbms Identifies the file to operate on as %DOCUMENTUM%\dba\config*repository_name*\dbpasswd.txt
You must include the -docbase argument if you include -rdbms.
 - file *file_name* Identifies the file on which to operate.
Do not include this argument if you include the -docbase argument.
 - encrypt *password* Defines the password to encrypt. If specified, the password is encrypted and written to the file identified in the -file argument. If unspecified, the utility encrypts the first line found in the file and writes it back to the file.

For example, executing the utility with the following command line replaces the database password used by the Content Server in the engineering repository to connect with the RDBMS:

```
dm_encrypt_password -docbase engineering -passphrase jdoe -rdbms
    -encrypt 2003password
```

The AEK location is not identified, so the utility reads the location from the DM_CRYPTO_FILE environment variable. The passphrase jdoe is used to decrypt the AEK. The utility encrypts the password 2003password and replaces the current RDBMS password in dbpasswd.txt with the newly encrypted password.

This next example identifies a user-defined file as the target of the operation.

```
dm_encrypt_password -passphrase jdoe
    -file C:\engineering\specification.enc -encrypt devpass
```

The AEK location is not identified, so the utility reads the location from the `DM_CRYPTO_FILE` environment variable. The password `jdoe` is used to decrypt the AEK. The utility encrypts the password `devpass` and writes the encrypted value to the file `C:\engineering\specification.enc`.

Limiting authentication attempts

Content Server supports an optional feature that allows you to limit the number of failed authentication attempts. You define the maximum number of allowed failures and if a user exceeds that number, he or she is inactivated in the repository. The feature is controlled by two attributes:

- `max_auth_attempt` in the doabase config object
- `failed_auth_attempt` in the user objects

The `max_auth_attempt` attribute defines the maximum number of allowed authentication failures. Failures that count toward the maximum include not only failed connection attempts, but also failures of any of the following methods on behalf of the user: Assume, Authenticate, Changepassword, and Signoff.

`max_auth_attempt` is set to 0 by default. To enable the feature, set the attribute to the maximum number of authentication failures you wish to allow.

The `failed_auth_attempt` attribute records how many authentication failures have occurred on behalf of a user. The attribute is set to 0 when a user is created and reset to 0 each time a user is successfully authenticated.

The `failed_auth_attempt` attribute is incremented by one for each failure. If `max_auth_attempt` is set to a positive number, Content Server inactivates the user when the value in the user's `failed_auth_attempt` attribute exceeds the value in `max_auth_attempt`. The inactivation is recorded in the server log file. The `failed_auth_attempt` attribute is reset to 0 when a system administrator reactivates the user by setting the `user_state` attribute to 0.

Note: If the maximum is lowered and a user has a value in `failed_auth_attempt` that exceeds the lowered maximum, his or her account remains active. If the next authentication attempt for that user succeeds, the attribute is reset to 0. If it does not succeed, the account is inactivated at that time.

If the user is the installation owner, inactivating the user shuts down the server automatically. For other users, inactivation puts restraints on any other open sessions the user may have. After the inactivation, those sessions allow only Assume and Disconnect operations. In a multi-repository distributed environment, inactivating a user in one repository has no effect on any open sessions the user may have in another repository.

To disable this feature for a particular user, set the `failed_auth_attempt` attribute to -1 for the user.

Note: The attribute is set to -1 for the installation owner by default when Content Server is installed.

Protecting Repository Objects

This chapter contains procedures for managing and using the features that provide security for the objects in a repository. For a description of all security features supported by Content Server, refer to [Chapter 4, Security Services](#), in *Content Server Fundamentals*. This chapter includes the following topics:

- [Overview of repository security, page 377](#)
- [Turning repository security on and off, page 380](#)
- [Turning folder security on and off, page 381](#)
- [Setting the default permission level for application-level control of SysObjects, page 383](#)
- [Object-level permissions, page 383](#)
- [Managing ACLs, page 387](#)
- [Table permits, page 404](#)
- [Auditing, page 406](#)
- [Implementing signature support, page 431](#)
- [Managing the encryption keys, page 443](#)
- [Managing the login ticket key, page 449](#)
- [Configuring login ticket use, page 450](#)
- [Configuring application access control token use, page 451](#)

Overview of repository security

In the Documentum system, every SysObject and SysObject subtype has associated security permissions that regulate access to the object. There are seven base access levels, ranging from no access to the permission to delete the object from the repository. There are also extended permissions that allow users to perform operations such as changing an object's owner or an object's location.

Object-level permissions are initially set when the object is created. If they are not explicitly set by the object's creator, then the system provides a default value. During the life of the object, its object-level permissions can be reset as many times as you like.

Enforcement of the permissions depends on whether security is turned on for the repository and the Access Control Lists (ACLs) associated with the object.

ACLs

An ACL is a list of access control entries that define access permissions and restrictions enforced for objects to which the ACL is applied. Every SysObject has one (and only one) associated ACL. When repository security is turned on, a user attempting to access a SysObject, such as a document, must own the object or be granted permission to access the object through an entry in the ACL. If neither condition is true, the user cannot access the object. (Note that a user always has at least Read access to the objects he or she owns.)

The basic server functionality supports ACLs that allow you to define a user's base and extended object level permissions. If the server was installed with a Trusted Content Services license, you can also create entries in an ACL that:

- Restrict a user's access to a specific permission level even if other entries in the ACL provide a higher level of access
- Require a user to belong to a specified group or groups before allowing the user access at his or her defined level
- Require a user to belong to at least one of a set of groups before allowing the user access at his or her defined level
- Include a user-defined permission recognized by applications

For complete information about ACLs, their implementation and how to create them, refer to [Managing ACLs, page 387](#).

Additional security options

In addition to the object-level permissions enforced using ACLs, Content Server supports the following security options.

Application-level control of SysObjects

Application-level control of SysObjects ensures that objects created by a particular application or client can only be modified by that application or other authorized applications. The feature is implemented using application codes. For more information refer to [Setting the default permission level for application-level control of SysObjects, page 383](#) in this manual and [Application-level control of SysObjects, page 86](#) in *Content Server Fundamentals*.

Dynamic groups

A dynamic group is a group, of any group class, whose list of members is considered a list of potential members. A dynamic group is created and populated with members like any other group. Whether or not a group is dynamic is part of the group's definition. It is recorded in the `is_dynamic` attribute and may be changed after the group is created.

When a session is started, whether Content Server treats a user in a dynamic group as an actual member is dependent on two factors:

- The default membership setting in the group object
- Whether the application from which the user is accessing the repository requests that the user be added or removed from the group

The `is_dynamic_default` attribute in a group object determines whether Content Server treats a user in a dynamic group's list of members as a group member or as a non-member. By default, the group setting directs Content Server to treat a user as a non-member of the group. If the application wants the user to be treated as a member, the application must issue an `IDfSession.addDynamicGroup` call. The alternative group setting directs Content Server to treat a user in the potential memberlist as members of the group unless the application issues an `IDfSession.removeDynamicGroup` call. (For information about setting this attribute, refer to [Changing the membership setting of a dynamic group, page 342.](#))

You can use dynamic groups to model role-based security. For example, suppose you define a dynamic group called `EngrMgrs`. Its default membership behavior is to assume that users are not members of the group. The group is granted the privileges to change ownership and change permissions. When a user in the group accesses the repository from a secure application, the application can issue the session call to add the user to the group. If the user accesses the repository from outside your firewall or from an unapproved application, no session call is issued and Content Server does not treat the user as a member of the group. The user cannot exercise the change ownership or change permissions permits through the group.

The session calls to add or remove users from a dynamic group can be audited. The events are named `dm_add_dynamic_group` and `dm_remove_dynamic_group`.

Folder security

Folder security is an optional level of security that is turned off or on using a attribute in the `docbase` config object. It not only checks permissions on the object, but also on one or more folders where the object is found. For more information, refer to [Turning folder security on and off, page 381.](#)

User privileges

A user privilege defines a user's capabilities within the repository. Each user has defined privileges. The default privilege is the lowest level, which gives a user no extra privileges. The other privileges must be granted specifically. There are two sets of user privileges, basic and extended. The highest basic privilege allows the user to read any object and to change the permissions on any object. The extended privileges define who can set, view, or delete audit trails. Like object-level permissions, user privileges are generally set when the user is created in the Documentum system and can be changed later if desired. User privileges are described in detail in [Chapter 9, Users and Groups](#).

Table permits

The table permits define access to RDBMS tables through Documentum. There are five levels of table permits, from none to delete. Table permits can be set for three user level: owner, group, and world. Table permits are described in [Table permits, page 404](#).

Turning repository security on and off

The `security_mode` attribute in the `docbase` config object controls whether object-level security is imposed on the repository. This attribute has two possible settings, listed in [Table 11-1, page 380](#).

Table 11-1. Repository security settings

Security setting	Description
none	There are no security checks—all SysObjects are considered public objects.
acl	Security is enforced using Access Control Lists (ACLs) defined for all SysObjects.

By default, the attribute is set to `acl`. To turn off object-level security, set `security_mode` to `none`. If the security mode is `none`, ACLs are not enforced. Neither, by extension, is folder security.

To reset the `security_mode` attribute through the API:

1. Connect to the repository as a user with Sysadmin or Superuser privileges.
2. Use the Set method to reset the `security_mode` attribute.

```
API>set, c, docbaseconfig, security_mode
```

```
SET>your_chosen_level
```

```
...
```

```
OK
```

3. Save your change.

```
API>save,c,docbase_config_id
```

You must use the docbase config object's object ID in the Save method. You cannot use the keyword docbaseconfig.

4. Reinitialize the server to make the change effective.

```
API>reinit,c,server_config_name
```

Turning folder security on and off

Folder security is a supplemental level of repository security. It must be explicitly turned on and only functions when repository security is also turned on. When folder security is turned on, the server performs the permission checks required by the repository security level and for some operations, also checks and applies permissions on the folder in which an object is stored or on the object's primary folder.

Folder security does not prevent users from working with objects in a folder. It provides an extra layer of security for operations that involve linking or unlinking, such as creating a new object, moving an object, deleting an object, and copying an object.

If folder security is turned on, the following security conditions are imposed in addition to any imposed by object-level security:

- Creating new objects requires Write permission on the folder or cabinet in which you want to store the new object.
- Linking an object to a folder or cabinet requires Write permission on the target folder or cabinet.
- Unlinking an object from a folder or cabinet requires Write permission on the affected folder or cabinet.
- Moving an object using the Link and Unlink methods requires Write permission on both the folder or cabinet from which you are unlinking and the folder or cabinet to which you are linking the object.
- Removing objects from the repository with Destroy and Prune requires Write permission for the object's primary folder.
- Prune cleans up a version tree, generally removing multiple versions of an object. You must have Write permission on the primary folder of each version removed by Prune.
- Copying an object using the Saveasnew API method or with a drag and drop operation or a key sequence in the Documentum clients requires Write permission

on all the folders or cabinets to which the new object will be linked. Copies created by Saveasnew have all the same links as the object that was copied.

For example, suppose you fetch DocA, which is linked to Folder1 and Folder2. Then you use Saveasnew to create a copy of DocA. This copy has the same links as the DocA. When you issue the Saveasnew method, the server checks the permissions on Folder1 and Folder2 because you created new links to the copy in those folders.

Changing folder security

You can change the folder security setting from Documentum Administrator or using the API.

To use API methods to change folder security:

1. Start an API session.

2. Fetch the doctype config object.

```
API>fetch,s0,doctypeconfig
...
OK
```

3. Set the folder_security attribute.

```
API>set,s0,l,folder_security
SET>T
...
OK
```

Or:

```
API>set,s0,l,folder_security
SET>F
...
OK
```

Note: The second argument is a lowercase L, which is an alias for the doctype config object returned in the previous step.

4. Save the change.

```
API>save,s0,l
...
OK
```

5. Reinitialize the server.

```
API>reinit,s0,server_config_name
```

The Reinit method uses the name of the server's server config object—not the doctype config object.

Setting the default permission level for application-level control of SysObjects

Application-level control of SysObjects ensures that objects controlled by a particular application or client can only be modified through that application or other authorized applications.

When a user accesses an object, Content Server determines whether the user's access is restricted because the object is controlled by a particular application. If so, the server checks to determine if the user is accessing the object through the controlling application or another application authorized to handle the object.

If a user isn't using the controlling application or another authorized application to access the object, then the user's access permission for the object is controlled by the setting in the `default_app_permit` attribute of the `docbase` config object or the user's permission as defined in the ACL, whichever is more restrictive.

The default value for `default_app_permit` is `Read`, meaning that users who attempt to access a controlled object through an unauthorized application are granted `Read` permission to the object unless the ACL for the object is more restrictive.

To change the `default_app_permit` setting, you must have `Sysadmin` or `Superuser` privileges. The lowest level to which this attribute can be set is `Browse`.

Object-level permissions

The object-level permissions consist of two sets of permissions: basic permissions and extended permissions. There are seven levels of basic permissions and five extended permissions.

Object-level permissions are defined in ACLs. (For information about setting permissions through ACL entries, refer to [Managing ACLs, page 387](#).)

Basic permissions

The access levels provided by the basic permissions are hierarchical. That is, the access capabilities at each level include those of all the levels below. For example, if a user is granted `Version` access, he or she can not only version the object but can also annotate it or examine its attributes. [Table 11-2, page 384](#) describes the seven access levels.

Table 11-2. Object-level permissions

Level	Permission	Description
1	None	No access is permitted.
2	Browse	The user can look at attribute values but not at associated content.
3	Read	The user can read content but not update.
4	Relate	The user can attach an annotation to the object. For information about how this permission level is applied to user-defined relationships, refer to Note on the Relate permit level, page 384 .
5	Version	The user can version the object.
6	Write	The user can write and update the object.
7	Delete	The user can delete the object.

[Table 11-3, page 384](#) presents a simple overview of the operations allowed at each permission level.

Table 11-3. Permitted operations for object-level pPermissions

Permit	Permitted actions					
	Fetch	Getfile	Annotate	Check-out/in	Save	Destroy
None						
Browse	X					
Read	X	X				
Relate	X	X	X			
Version	X	X	X	X		
Write	X	X	X	X	X	
Delete	X	X	X	X	X	X

Note on the Relate permit level

Documentum allows users to create user-defined relationships between two objects. The two objects involved in the relationship are termed the parent and child. That is, the

user must, at the time that he or she creates the relationship, name one of the objects as the parent and one as the child.

The system uses these designations in conjunction with the defined security level for the relationship to enforce security. There are four security levels for user-defined relationships: system, parent, child, and none.

- If the system security level is defined, then you must have Sysadmin or Superuser privileges to create, modify, or destroy any relationships of that type.
- If the security level is parent, you must have at least Relate permission for the object defined as the parent to create, modify, or destroy the relationship.
- If the security level is child, you must have at least Relate permission for the object defined as the child to create, modify, or destroy the relationship.
- If the security level is none, there are no permissions necessary to create, modify, or destroy the relationship.

Extended permissions

The access levels provided by the extended permissions are not hierarchical. These permissions work together with the basic permissions.

[Table 11-4, page 385](#) describes the extended permissions.

Table 11-4. Extended object-level permissions

Permission	Description
change_location	<p>In conjunction with the appropriate base permission level, allows the user to move an object from one folder to another.</p> <p>All users having at least Browse permission on an object are granted Change Location permission by default for that object.</p> <p>Note: Browse permission is not adequate to move an object. For a description of privileges necessary to link or unlink an object, refer to the Link and Unlink method descriptions in the <i>Content Server API Reference Manual</i>.</p>
change_owner	The user can change the owner of the object.
change_permit	The user can change the basic permissions of the object.

Permission	Description
change_state	The user can change the document lifecycle state of the object.
delete_object	The user can delete the object. The delete object extended permission is not equivalent to the base Delete permission. Delete Object extended permission does not grant Browse, Read, Relate, Version, or Write permission.
execute_proc	The user can run the external procedure associated with the object. All users having at least Browse permission on an object are granted execute_proc permission by default for that object.

Viewing extended permissions

Table 11-5, page 386 lists the computed attributes that return information about the extended permissions.

Table 11-5. The extended permission computed attributes

Attribute	Single/ Repeating	Description
_allow_execute_proc	S	A Boolean value indicating whether the user has the Execute Procedure permission
_allow_change_location	S	A Boolean value indicating whether the user has the Change Location permission
_allow_change_state	S	A Boolean value indicating whether the user has the Change State permission
_allow_change_permit	S	A Boolean value indicating whether the user has the Change Permission permission
_allow_change_owner	S	A Boolean value indicating whether the user has the Change Ownership permission

Attribute	Single/ Repeating	Description
<code>_accessor_xpermit</code>	R	The integer value of the extended permissions assigned to each user or group returned by <code>_accessor_name</code> . The values, expressed as integers, are associated with the user or group at the corresponding index level. For example, the permit level at <code>_accessor_xpermit[4]</code> is assigned to the user or group specified by <code>_accessor_name[4]</code> .
<code>_accessor_xpermit_names</code>	R	A list of the extended permissions, in string form, assigned to each user or group returned by <code>_accessor_name</code>
<code>_xpermit</code>	R	The integer value of the extended permissions that the current user or the specified user has on the object
<code>_xpermit_names</code>	S	The list of extended permissions, in string form, that the current user or the specified user has on the object.
<code>_xpermit_list</code>	S	A full list of the extended permissions, in string form, currently supported by the server. Note that this computed attribute returns the same list regardless of the object ID.

Managing ACLs

This section describes how access control lists (ACLs) are implemented, how they behave in the Documentum system, and how to work with them. The following topics are included:

- [The ACL object type, page 388](#)
- [How ACL entries are evaluated, page 392](#)
- [Disabling ACL restrictive entries, page 396](#)
- [External and internal ACLs, page 396](#)
- [System, public, and private ACLs, page 397](#)
- [Template ACLs, page 398](#)

- [Creating ACLs, page 398](#)
- [How ACLs and objects are connected, page 399](#)
- [The default ACLs, page 400](#)
- [Modifying an ACL, page 402](#)
- [Destroying an ACL, page 403](#)

The ACL object type

Access control lists are stored as persistent objects of type `dm_acl`. The single-valued attributes of an ACL object contain information about the ACL, such as its name and its owner. The repeating attributes define the access control entries. An ACL can contain any number of access control entries.

Although ACLs are persistent objects having an object ID, they are not `SysObjects`. You cannot version an ACL. If you modify an ACL, the server either overwrites the ACL with the changes or copies the ACL and changes the copy. Which option it chooses depends on whether you reference the ACL directly to make the changes or reference an object that uses the ACL.

Access control entries

The access control entries are defined in the repeating attributes in an ACL. The values at one index position across the attributes represent one access control entry. Each entry defines one of the following:

- An access permission, extended permission, or both, for a user or group
- An access restriction, extended restriction, or both, for a user or group
- A required group
- A required group set
- An application permission
- An application restriction

Note: Access permissions and extended permissions are supported by the basic Content Server functionality. Creating restricting entries, required group entries, required group set entries, or application entries requires a Trusted Content Services license.

An entry's permit type identifies what is defined in the entry. The identification is recorded in the `r_permit_type` attribute as an integer value. Access and extended permissions for a given user or group are always stored in the same entry. The `permit_type` for that entry is set to the integer value representing an access permission. Similarly, access restrictions and extended restrictions for a given user or group are

always stored in the same entry, and the `permit_type` for that entry is set to the integer value representing an access restriction.

AccessPermission and ExtendedPermission entries

An access permission entry defines a base object-level permission for a user or group. The base object-level permissions are None, Browse, Read, Relate, Version, Write, and Delete. (Refer to the *Content Server Administration Guide* for a full description of the base object-level permissions.) The permit type of an access permission entry is `AccessPermit`.

An extended permission entry defines an extended object-level permission for a user or group. The extended object-level permissions are: `change_location`, `change_owner`, `change_state`, `change_permit`, `delete_object`, and `execute_proc`. The permit type for an extended permission is `ExtendedPermit`.

In the ACL, both access permissions and extended permissions are stored in the same entry, whose `permit_type` is set to `AccessPermit`. However, when you grant or revoke these permissions, you must specify whether you are granting or revoking an `AccessPermit` or `ExtendedPermit`.

AccessRestriction and ExtendedRestriction entries

Restriction entries restrict a user or group's access. Creating entries of these types requires a Trusted Content Services license.

AccessRestriction entries

An access restriction entry removes the right to the base object-level permission level specified in the entry. The user or group members have access at the level up to the specified restriction. Access restriction entries are useful when you want give a group a particular base object-level permission, but restrict access for individual members or a subgroup of members. For example, suppose that Olivia is a member of the `ProjTeam` group and that an ACL has the following entries:

```
ACCESSOR_NAME: ProjTeam
PERMIT_TYPE: AccessPermit
PERMIT_LEVEL: delete

ACCESSOR_NAME: Olivia
PERMIT_TYPE: AccessRestriction
PERMIT_LEVEL: version
```

All members of the `ProjTeam` except Olivia have permission to delete objects governed by this ACL. Olivia's permission is restricted to browsing, reading, or annotating the

objects even though she is a member of the ProjTeam. She cannot version the objects, or write or delete them.

The permit type for an access restriction entry is `AccessRestriction`.

ExtendedRestriction entries

An extended restriction entry restricts a user or the members of a specified group from exercising the specified extended object-level permission. For example, suppose that an ACL has the following entries and that HortenseJ is a member of the ProjTeam group:

```
ACCESSOR_NAME: ProjTeam_grp
PERMIT_TYPE: ExtendedPermit
PERMIT_LEVEL: change_owner,change_permit
```

```
ACCESSOR_NAME: HortenseJ
PERMIT_TYPE: ExtendedRestriction
PERMIT_LEVEL: change_permit
```

In this example, HortenseJ is restricted from the `change_permit` permission even though she is a member of a group whose members are granted this permission. She cannot change the permissions of any object governed by this ACL.

The permit type of an extended restriction is `ExtendedRestriction`.

Storage in the ACL

In the ACL, both access restriction entries and extended restriction entries for a particular user or group are stored in the same ACL entry, with the `permit_type` set to `AccessRestriction`. However, when you grant or revoke an access restriction or extended restriction, you must specify whether you are granting or revoking an `AccessRestriction` or `ExtendedRestriction`.

ApplicationPermission entries

You must have installed Content Server with a Trusted Content Services license to create application permission entries.

An application permission entry specifies a user-defined permission level that is recognized by one or more user-written applications. Content Server does not recognize permission levels identified in application permission entries. The user application must recognize and enforce any application permissions.

The permit type for an application permit entry is `ApplicationPermit`.

ApplicationRestriction entries

You must have installed Content Server with a Trusted Content Services license to create application restriction entries.

An application restriction entry identifies a user or group that is not allowed to exercise the privileges associated with a user-defined application permission level. For example, suppose that HarleyJ is a member of the ProjTeam group and that an ACL has the following entries:

```
ACCESSOR_NAME:ProjTeam_grp
PERMIT_TYPE:ApplicationPermit
APPLICATION_PERMIT: shred

ACCESSOR_NAME: HarleyJ
PERMIT_TYPE:ApplicationRestriction
APPLICATION_PERMIT: shred
```

This ACL grants shred permission to the ProjTeam group, but restricts HarleyJ from that permission even though he is a member of the ProjTeam group.

Application restriction entries, like application permissions, are not recognized by Content Server, but must be enforced by the user-defined applications that enforce the application permissions.

The permit type of an application restriction is ApplicationRestriction.

RequiredGroup entries

You must have installed Content Server with a Trusted Content Services license to create required group entries.

A required group entry requires a user requesting access to an object governed by the ACL to be a member of the group identified in the entry. For example, suppose an ACL has the following entries:

```
ACCESSOR_NAME: GaryG
PERMIT_TYPE: AccessPermit
PERMIT:Delete

ACCESSOR_NAME:ProjTeam
PERMIT_TYPE:RequiredGroup
PERMIT:NULL (not applicable)

ACCESSOR_NAME:Engr
PERMIT_TYPE:RequiredGroup
PERMIT:NULL (not applicable)
```

When GaryG attempts to access a document governed by this ACL, Content Server checks to determine whether he is a member of both the ProjTeam and Engr groups before allowing access. GaryG must belong to both groups. If he is not a member of both

groups, the server does not allow him to access the document. The only exception to this is a superuser. A superuser is not required to be a member of any required group to access a document.

The permit type for a required group entry is `RequiredGroup`.

RequiredGroupSet entries

You must have installed Content Server with a Trusted Content Services license to create required group set entries.

A required group set entry requires a user requesting access to an object governed by the ACL to be a member of at least one group in the set of groups. An ACL that enforces a required group set typically has multiple required group set entries. Each entry identifies one group in the set. The user must belong to at least one of the groups identified by the required group set entries in the ACL.

For example, suppose an ACL has the following entries:

```
ACCESSOR_NAME: HollyH
PERMIT_TYPE: AccessPermit
PERMIT:Delete
```

```
ACCESSOR_NAME: ProjTeam
PERMIT_TYPE: RequiredGroupSet
PERMIT:NULL (not applicable)
```

```
ACCESSOR_NAME: Engr
PERMIT_TYPE: RequiredGroupSet
PERMIT:NULL (not applicable)
```

When HollyH tries to access an object governed by this ACL, Content Server determines whether she is a member of either the ProjTeam or Engr group. She must be a member of one of these groups to be given access to the object. The only exceptions to this are superusers. A superuser is not required to be a member of any required group set to access a document.

Because required group set entries in an ACL are all considered to belong to that same set, each ACL can have only one required group set.

The permit type for a required group set entry is `RequiredGroupSet`.

How ACL entries are evaluated

This section describes how Content Server uses the entries in an ACL to evaluate access for users.

Note: A Content Server evaluates all entries in an ACL regardless of whether the server was installed with a Trusted Content Services (TCS) license or not. Lack of a TCS license only restricts the ability to create or modify the ACL entries. Lack of TCS license does not restrict the ability to evaluate such entries when present in an ACL.

Evaluation for non-owners and non-superusers

When a user who is not an object's owner or not a superuser requests access to a SysObject, Content Server evaluates the entries in the object's ACL in the following manner:

1. The server checks that there is an AccessPermit type entry that gives the user the requested base or extended access level (browse, read, write, and so forth)

Note: Users are always given Read access if the user owns the document regardless of whether there is an explicit entry granting Read access or not.

2. The server next checks that there are no AccessRestriction type entries that deny the user access at the requested level.

A restricting entry, if present, may restrict the user specifically or may restrict access for a group to which the user belongs.

3. If there are RequiredGroup type entries, the server checks that the user is a member of each specified group.
4. If there are RequiredGroupSet type entries, the server checks that the user is a member of at least one of the groups specified in the set.

If the user has the required permission, with no access restrictions, and is a member of any required groups or a required group set, the user is granted access at the requested level.

How access is evaluated for object owners and superusers

Content Server uses a different algorithm to evaluate access for owners of an object or superusers than it uses for users who do not own the object or are not superusers.

Access evaluation for an object's owner

Content Server uses the following algorithm to determine an owner's access permissions:

1. Checks that the owner belongs to any required groups or a required group set.
If the owner doesn't belong to the required groups or group set, then the owner is allowed only Read permission as his or her default base permission and the owner is granted none of the extended permissions.
2. Determines what base and extended permissions are granted to the owner through entries for `dm_owner`, the owner specifically (by name) or through group membership.
3. Then, applies any restricting entries for `dm_owner`, the owner specifically (by name), or any groups to which the owner belongs.
4. The result constitutes the owner's base and extended permissions.
 - If there are no restrictions on the base permissions of the owner and the `dm_owner` entry does not specify a lower level, the owner has Delete permission by default.
 - If there are restrictions on the base permission of the owner, the owner has the permission level allowed by the restrictions. Note that the owner cannot be restricted below Browse permission. An owner cannot be restricted to None permission. The owner will always have at least Browse permission.
 - If there are no restrictions on the user's extended permissions, the user has, at minimum, all extended permissions except `delete_object` by default. The user may also have `delete_object` if that permission was granted to `dm_owner`, the user by name, or through a group to which the user belongs.
 - If there are restrictions on the user's extended permissions, then the user's extended permissions are those remaining after the restrictions are applied.

To illustrate how extended permissions are evaluated, here is an example. Suppose that Jorge is a member of the `QA_grp` and that Jorge opens a document he owns whose ACL entries are:

```
ACCESSOR NAME: dm_owner
ACCESS PERMIT: delete
EXTENDED PERMIT: delete object
```

```
ACCESSOR NAME: QA_grp
ACCESS PERMIT: version
EXTENDED PERMIT: change location, execute procedure
```

```
ACCESSOR NAME: QA_grp
EXTENDED RESTRICTION: change permit
```

By default, as owner, Jorge has all extended permissions except `delete_object`. Working from this default, Content Server adds the extended permissions granted to `dm_owner` and subtracts any extended restrictions that apply to Jorge or any groups to which he belongs. In this example, Jorge gains the `delete_object` extended permission, but loses the `change_permit` permission. His result set of extended permissions is:

- change location
- change owner
- change state
- execute procedure
- delete object

Evaluating a Superuser's permissions

When Content Server evaluates a superuser's access to an object, the server does not apply `AccessRestriction`, `ExtendedRestriction`, `RequiredGroup`, or `RequiredGroupSet` entries to a superuser. A superuser's base permission is determined by evaluating the `AccessPermit` entries for the user, for `dm_owner`, and for any groups to which the user belongs. The superuser is granted the least restrictive permission among those entries. If that permission is less than `Read`, it is ignored and the superuser has `Read` permission by default.

A superuser's extended permissions are all extended permits other than `delete_object` plus any granted to `dm_owner`, the superuser by name, or to any groups to which the superuser belongs. This means that the superuser's extended permissions may include `delete_object` if that permit is explicitly granted to `dm_owner`, the superuser by name, or to groups to which the superuser belongs.

Resolving multiple entries for a user

A user can have an entry as an individual and as a member of one or more groups that have access. In such cases, the user's basic permission is the least restrictive of the entries applicable to the user, and the user's extended permissions are all applicable extended permissions.

For example, suppose that `johnpq` is a member of `engr` and `qa_test` groups. An ACL is created with the following entries:

- User `johnpq` has an entry that gives him `Delete` permission.
- The `engr` group has entries that give `engr` `Write` permission and the `Change Location` and `Change Permission` extended permissions.
- The `qa_test` group has entries that give `qa_test` `Version` permission and the `Execute Procedure` extended permission.

When the server evaluates the permissions given to `johnpq` by this ACL, it determines that `johnpq` has `Delete` permission on objects (the least restrictive) and `Change Location`, `Change Permission`, and `Execute Procedure` permissions (the combined set).

Disabling ACL restrictive entries



Caution: If a repository is licensed for Collaborative Services, do not disable the use of restrictive entries. Collaborative Services features do not work when restrictive entries are disabled.

You can turn off the use of the following types of ACL access control entries that restrict access:

- AccessRestriction
- ExtendedRestriction
- RequiredGroup
- RequiredGroupSet

Use of these entries can be disabled at the repository level by setting the `macl_security_disabled` attribute to `TRUE`. When that attribute is `TRUE`, only base and extended access permissions in an ACL are used to determine a user's access to an object.

Disabling the evaluation of restrictive entries does not affect the ability to add such entries to an ACL. It only stops enforcement of those entries.

Use Documentum Administrator to change the attribute's setting. When Content Server is installed with a Trusted Content Services license, the attribute is set to `FALSE` by default, meaning all ACL entries are used to evaluate access.

External and internal ACLs

ACLs are either external or internal ACLs, depending on whether they are created and managed externally by a user or internally by Content Server.

External ACLs are created explicitly by users and managed (modified or deleted) by the user who created them. Typically, they are created using Documentum Administrator. You can also create them using the DFC.

Internal ACLs are created by the server as a response whenever a user sets or modifies an object's permissions by directly referencing the object. This can occur in the following situations:

- A user creates a `SysObject` and sets permissions for that object but does not explicitly associate an ACL with the object.

In this situation, the server creates an internal ACL based on the default ACL defined at the server level (in the `default_acl` attribute of the server's server config object). It copies that default ACL, makes the specified changes, and then assigns the copy to the document. The copy is an internal ACL. ([The default ACLs, page 400](#), contains information about the default ACLs.)

- A user creates a SysObject, associates an ACL with the object, and then modifies the access control entries in the ACL.

Content Server copies the ACL that the user associated with the object, makes the changes in the copy, and then assigns the copy to the object, replacing the first ACL. The copy is an internal ACL.

- A user creates a document, does not assign an ACL to the document and issues a Useacl method with the none option to tell the server not to assign a default ACL, and then uses Grant to define permissions for the document.

Note: Important: When an internal ACL is created in this manner, the server does not automatically give the object's owner or the world access to the object. In these cases, the server sets the access for dm_owner and dm_world to 1 (None). Although the server automatically gives an object's owner Read access to the object, you must grant any higher access to the owner explicitly. You must also explicitly grant access to dm_world in such cases.

- The user fetches an existing document and modifies its permissions

When this occurs, the server copies the document's current ACL, makes the specified changes in the copy, and then assigns the copy to the document. The new copy is an internal ACL. In this way, the changes only apply to the specified document.

Internal ACLs are managed by the server. (Internal ACLs are called custom ACLs in *Content Server Fundamentals*.)

ACL names

An ACL name can be any valid name up to 32 characters long. (Refer to [Identifiers](#), page 30, in the *EMC Documentum Object Reference Manual* for naming rules.)

Internal ACLs are named by the server. The server assigns a unique name in the format:

`dm_acl_object_id`

For example: dm_450000230006453f

If a user does not specify a name for an external ACL when he or she creates it, the server assigns a name. The format is the same as for internal ACLs:

`dm_acl_object_id`.

System, public, and private ACLs

Every ACL, whether external or internal, is either a public ACL, a system ACL, or a private ACL.

Public ACLs are ACLs that are available for anyone in the repository to use. Only the owner of a public ACL or a user with Sysadmin or Superuser privileges can modify or delete the ACL.

System ACLs are a subset of public ACLs. They are public ACLs that are owned by the repository owner.

Private ACLs are ACLs that are available only to the users who create them. Any user in the repository, except the repository owner, can create private ACLs. A repository owner has no private ACLs. All ACLs created by the repository owner are system ACLs. Only the owner of a private ACL, or a superuser, can modify or delete the ACL.

After an ACL is created, only a superuser can change its name or owner. However, because ACLs are referenced internally by their names and owners, it is strongly recommended that names and owners not be changed. If you do so, you must also change the references on all objects associated with the ACL.

Template ACLs

A template ACL is an ACL that has the `acl_class` attribute set to 1. Typically, template ACLs have one or more `r_accessor_name` values set to alias specifications. When used in ACLs, aliases are place-holders for user and group names. Use template ACLs with aliases in applications that are intended to run in a variety of contexts. The aliases ensure that the `r_accessor_name` values are always appropriate for the context of the application.

When you assign a template ACL to an object, the server copies the template, resolves the aliases and replaces them in the copy with the real names, and assigns the copy to the object. The copy is always instantiated as a system-level ACL with an `acl_class` attribute value of 2. Users cannot change the copy. If a template ACL or the alias set used to resolve the template's aliases is modified, the server automatically updates the instantiated copies also.

[Creating ACLs, page 398](#), contains information about creating an ACL template.

Creating ACLs

Use Documentum Administrator to create an ACL. You can use DFC to create an ACL also, if you need to create one programmatically. You cannot use DQL to create an ACL.

You must have installed Content Server with a Trusted Content Services license to create entries of the following types in the ACL:

- AccessRestriction and Extended Restriction
- RequiredGroup and RequiredGroupSet

- ApplicationPermit and ApplicationRestriction

You must have Superuser or Sysadmin privileges or be the repository owner to create a system ACL.

Any user in a repository except the repository owner can create a private or public ACL. ACLs created by the repository owner are always system ACLs.

Any user can create a template ACL. To create a template ACL, use Documentum Application Builder. You can also create one programmatically. If you do so, you must set the `acl_class` attribute to 1 and set the `r_accessor_name` values to aliases.

For information about aliases and the format of an alias specification, refer to [Appendix A, Aliases](#) in the *Content Server Fundamentals*. The appendix also contains a description of how the server resolves an alias.

How ACLs and objects are connected

Every object with an ACL has two attributes that define which ACL is associated with that object. The two attributes are:

- `acl_domain`

This attribute contains the name of the owner of the ACL associated with the object. For private ACLs, this is the name of the user who created the ACL. For system ACLs, this is the name of the repository owner.

- `acl_name`

This attribute contains the name of the ACL. This will either be the name specified by the user who created the ACL or a string with the format:

`dm_acl_object_id`

For example: `dm_4500025400002e7f`

All internal ACLs have names that use their object IDs. External ACLs are generally named by the user who creates them. If the user does not name them, the server will name them using the same format that it uses for internal ACL names.

For SysObjects, these two attributes define the ACL that contains the access permission for the object. For user and type definitions, the attributes identify the associated default USER and TYPE ACLs. (For types, these attributes are found in the type's associated type info object. Type info objects contain all the non-structural information about a type. There is one type info object for each type in Documentum.)

The default ACLs

Businesses often want to define a default set of ACLs for users. A business may do this as a business rule, to provide standard controls for document access. Default ACLs may also be provided as a service to end users, to make it easy for them to create new objects without worrying about security settings.

Documentum lets you define three possible default ACLs:

- Folder

A folder ACL is an ACL that is associated with a folder or cabinet. A folder ACL has two purposes:

- It can be used as the default ACL for any object that has the folder or cabinet as its primary folder.

Note: The primary cabinet or folder is recorded in an object's `i_folder_id[0]` value. If an object is placed directly in a cabinet, instead of in a folder, then the cabinet's object ID is found in `i_folder_id[0]`, and the cabinet is both the object's primary storage location and its primary folder.

- It may be used to determine access rights to the folder or cabinet if folder security is turned on for the repository.

A folder or cabinet's default ACL is recorded in the object's `acl_name` and `acl_domain` attributes.

- User

A user ACL is an ACL that is associated with a user object. When the repository owner or a user with Sysadmin or Superuser privileges creates a user in the repository, he or she also assigns or defines an ACL for the user. This ACL can be used as the ACL for any object created by the user. Because user objects are not subtypes of SysObject, the ACL is not used to enforce any kind of security on the user; a user's ACL can only be used as a default ACL.

A user's default ACL is recorded in the user's `acl_name` and `acl_domain` attributes.

- Type

A type ACL is an ACL that is associated with the type definition of a SysObject or SysObject subtype. For example, you can define a default ACL for all objects of type `dm_document`. When a user creates a new document, he or she can assign the type's default ACL to the new object. (If the type has no default ACL, the system will search the type hierarchy until it finds a type with a default ACL to assign.)

The type's default ACL is defined in the type's type info object. Each object type has an associated type info object that contains all the non-structural information about the type.

There are no ACLS defined for object types by default. If you wish to use this option as the default ACL, you must modify the object types to add the default ACL to each type definition. Use ALTER TYPE to add the ACL to the object type definition. When this default is being used, if a particular type has no defined ACL, Content Server will traverse an object's type tree to find an ACL. Consequently, if you are using this option, it is recommended that the dm_sysobject type, at the least, has an ACL defined for the type.

Creating default ACLs

A default ACL has no special properties. To create one, you simply create an ACL that has the access control entries you wish and then associate the ACL with the appropriate folder, user, or type info object.

It is recommended that you use Documentum Administrator to assign default ACLs to users, cabinets, folders, and object types.



Caution: When you assign an ACL to a folder or type definition, be sure to assign a system ACL if the folder or type is publicly accessible. If you assign a private ACL as the folder or type default, when a user who does not own the ACL attempts to assign it to an object, the operation will fail.

Assigning a default ACL to an object

There are two ways to assign a user, type, or folder ACL to an object. First, a user can assign one explicitly with a Useacl method. Alternatively, if a user does not assign any permissions to a new object, the server will automatically assign a one of the three based on the value in the default_acl attribute of the server's server config object.

Note: If the new object is a cabinet and the user does not explicitly assign an ACL, the server assigns the user's default ACL to the cabinet rather than the ACL specified in the server's default_acl attribute.

For details about using Useacl, refer to [Assigning ACLs, page 147](#), in *Content Server Fundamentals*. For information about setting the default_acl attribute, refer to [Identifying the default ACL for use, page 401](#).

Identifying the default ACL for use

If a user does not identify an ACL for an object or does not explicitly that the object should not have a default ACL, the server attempts to assign the new object a default

ACL. To identify which ACL to assign as the default, the server uses the value in the `default_acl` attribute of its server config object. This attribute contains an integer value that represents one of three candidate ACLs: 1, which is the folder ACL; 2, which is the type ACL; or 3, which is the user ACL. A value of 4 means that there is no default ACL.

When you install Content Server, the `default_acl` attribute is set to 3, for the user ACL. This means that whenever a user creates an object and does not explicitly assign it an ACL or grant it permissions, the server assigns the default ACL associated with the user's `dm_user` object to the new object.

You can change the `default_acl` attribute using Documentum Administrator or the API.

Modifying an ACL

As the owner of system ACLs, you may have to modify their access control entries. For example, as users join or leave the company, you may have to make appropriate changes in the system ACLs. To make these changes, you can use Documentum Administrator or Grant and Revoke methods. The Grant method adds an entry to an ACL. The Revoke method removes entries for a particular user or group, or removes an extended permission.

You must have installed Content Server with a Trusted Content Services license to modify, add, or delete the following types of entries:

- AccessRestriction and Extended Restriction
- RequiredGroup and RequiredGroupSet
- ApplicationPermit and ApplicationRestriction

Adding entries

You can use Documentum Administrator or the API to add an entry to a system ACL. You can only change an ACL if you own the ACL or have Superuser privileges. The changes you make affect every object that uses the ACL.

If you use Documentum Administrator, refer to the online help for instructions. If you use the API, use a Grant method. For the syntax and instructions on its use, refer to [Grant, page 258](#) in the *Content Server API Method Reference*.

Removing entries

You can use Documentum Administrator or the API to remove access control entries. You must own the ACL or have Superuser privileges to remove entries from an ACL.

If you use the API, use a Revoke method. For the syntax and instructions on its use, refer to [Revoke, page 400](#) in the *Content Server API Method Reference*.

Destroying an ACL

Removing ACLs from the repository must be done explicitly. The server does not automatically remove ACLs that are no longer referenced by any object.

When you delete an ACL, Content Server also deletes any registry objects that reference the ACL as the subject of an audit or event notification request.

Removing unreferenced external ACLs

When you decide that you no longer want to use an external ACL and it is not referenced by any objects in the repository, you can remove it from the repository by using Documentum Administrator or by using the Destroy method.

To destroy an external ACL, the following conditions must be true:

- You must be either the owner of the ACL or have Superuser privileges.
- The ACL cannot be referenced by any objects in the repository.

To use a Destroy method, you must know the object ID of the ACL object. ([Destroy, page 186](#) describes how to use Destroy.) If you do not have the object IDs for the ACLs you want to remove, use a DQL SELECT statement to retrieve them. For example:

```
SELECT "r_object_id", "object_name" FROM "dm_acl"  
WHERE "owner_name" = 'name_of_acl_owner'
```

Removing unreferenced internal ACLs

To remove unreferenced internal ACLs from the repository, use the dmclean utility. You must have Sysadmin or Superuser user privileges to run dmclean. You can run dmclean using:

- Documentum Administrator
- A DQL EXECUTE statement

The syntax is:

```
EXECUTE do_method
WITH method = 'dmclean',
arguments = '-no_content -no_note -no_wf_template'
```

- An Apply method

The syntax is:

```
apply, session, NULL, DO_METHOD, METHOD, S, dmclean,
ARGUMENTS, S, '-no_content -no_note -no_wf_template'
```

- From the operating system prompt

```
dmclean -docbase_name repository -init_file init_file_name
-no_content -no_note -no_wf_template
```

- where *repository* is the name of the repository that contains the ACLs and *init_file_name* is the name of the server.ini start up file for the repository's server.

The `dmclean` utility removes all internal ACLs that are not referenced by any object. Specifying `-no_content`, `-no_note`, and `-no_wf_template` ensures that only unreferenced ACLs are removed from the repository. If these arguments are not included, the utility also scans the repository for orphaned content objects and files, orphaned note objects, and orphaned SendToDistributed workflow templates. Orphaned note objects and workflow templates are removed and a script is generated for removing orphaned content objects and files.

Table permits

The table permits work in conjunction with object-level permissions to control access to the RDBMS tables that underlie registered tables. To query an RDBMS table underlying a registered table, users must have:

- At least Browse access for the `dm_registered` object representing the RDBMS table
- The appropriate table permit for the desired operation

The only exception to the above constraints applies to Superusers. Users with Superuser privileges can issue SELECT queries against any RDBMS table, including unregistered tables.

There are five levels of table permits, described in [Table 11–6, page 404](#).

Table 11-6. Table permits

Level	Permit	Description
0	None	No access is permitted
1	Select	The user can retrieve data from the table.

Level	Permit	Description
2	Update	The user can update existing data in the table.
4	Insert	The user can insert new data into the table.
8	Delete	The user can delete rows from the table.

The permits are not hierarchical. For example, assigning the permit to insert does not confer the permit to update. To assign more than one permit, you add together the integers representing the permits you want to assign and set the appropriate attribute to the total. For example, the following statement sets the group permit on a document to both insert and update:

```
dmAPISet ("set,s0,0900002e4311125e,group_table_permit","6")
```

Setting table permits

The table permits are defined for owner, group, and world access through attributes of the registered object representing the registered table. (ACLs are not used.) The attributes are:

- owner_table_permit
- group_table_permit
- world_table_permit

The owner is the user who created the registered object for the RDBMS table. The group represents all users who belong to the group defined for the registered object. The world is all users who are not the owner and not part of the registered object's group.

When you create a registered table, the system sets the default permit level to Select for the owner. The default permit level for the group and world levels is None. You can change the defaults by resetting the attributes.

You can set these attributes at any time after you create the registered table. The attributes have an integer datatype. When you set them, you define the integer value that corresponds to the permit level you want to assign. For example, the following method sets the group permit on a registered object to Update:

```
dmAPISet ("set,s0,reg_tbl_objid,group_table_permit","2")
```

This method gives the world both the Select and Update permits:

```
dmAPISet ("set,s0,reg_tbl_objid,world_table_permit","3")
```

Table permits and object-level permissions

You must have at least Browse permission for the dm_registered object that represents an RDBMS table in order to access the RDBMS table at any table permit level. However, other than this exception, table permits and object-level permissions do not affect each other.

It is possible to have only Read access to a dm_registered object and have a Delete permit for its underlying RDBMS table. Similarly, you might have Write access to the dm_registered object, but only a Select permit for its underlying RDBMS table.

Table permits and dump and load operations

Table permit values are carried over to the target repository when a registered table is dumped and loaded.

Auditing

Auditing is a security feature that allows you to monitor events that occur in a repository or application. Events are operations performed on objects in a repository or something that happens in an application.

Auditing an event creates an audit trail, a history in the repository of the occurrence of the event. Creating an audit trail is a useful way to prove compliance with a business rule. You can also use the information in an audit trail to:

- Analyze patterns of access to objects
- Monitor when critical documents change or when the status of a critical document changes
- Monitor the activity of specific users

There are many ways to conduct auditing. For example, you can audit:

- All occurrences of a particular event on a given object or given object type
- All occurrences of a particular event in the repository, regardless of the object to which it occurs
- All workflow-related events in the repository
- All occurrences of a particular workflow event for all workflows started from a given process definition
- All executions of a particular job
- All events in the repository

Depending on the particular event and its target, you can also choose to audit an event only when the target is controlled by a particular application, attached to a particular lifecycle, or in a particular state in a particular lifecycle.

What events are auditable

There are two kinds of auditable events: *system events* and *application events*. System events are events that Content Server recognizes and can audit automatically. For example, checking in a document can be an audited system event. Application events are events that are recognized and audited by client applications. For example, a user opening a particular dialog can be an audited application event. (A complete list of auditable system events is in [Appendix B, System Events](#), in the *Content Server API Reference Manual*.)

Auditing attributes

When you initiate auditing for an event that targets an object or object type, you can identify attributes of the object whose values you wish to audit also. For example, suppose you have a subtype called `vendor_info` with an attribute named `vendor_number`. You want to audit all changes to current vendors. To do so, register the `dm_checkin` event for auditing whenever a `vendor_info` object is the target of the checkin and identify the `vendor_number` as an audited attribute. Each time a user checks in a `vendor_info` document, Content Server creates an audit trail entry and includes the `vendor_number` attribute name and value in the entry, providing you with information that identifies which vendor has changed information.

For system events, the values of audited attributes stored in the audit trail entries are the values of the attribute after the audited object is saved to the repository. If the object is versioned, the value is the value of the attribute in the new version. For example, suppose the `vendor_info` object type has an attribute called `vendor_address`. If you are auditing that attribute for checkin events, the new address appears in the audit trail entry. Similarly, if you audit the `r_object_id` attribute when checkin events are audited, the audit trail entry records the object ID of the new version of the document as the audited value of the `r_object_id` attribute.

For application events, Content Server also records audited attribute names and values in the audit trail entry if you use `Createaudit` to create the audit trail entry. If you use `Create`, `Set`, and `Save` methods to create the entry, the application is responsible for recording the names and values of audited attributes in the audit trail entry.

Audit trails

An audit trail is the history of an audited event. Each occurrence of an audited event is recorded in one entry in an audit trail. Audit trail entries are stored in the repository as persistent objects. Depending on the event, the objects are `dm_audittrail`, `dm_audittrail_acl`, or `dm_audittrail_group` objects. Audit trail entries store pertinent information about the events, such as when the events occurred, what objects were involved, and who performed the actions.

Audited attributes are recorded in the `attribute_list` attribute of a `dm_audittrail` object. If the list of audited attributes and their values is too large for the attribute, the overflow is stored in a `dmi_audittrail_attrs` object and the object ID of the `dmi_audittrail_attrs` object is stored in the `attribute_list_id` attribute of the `dm_audittrail` object.

Default auditing

Content Server audits the following events by default:

- All executions of an Audit or Unaudit method
The event names are `dm_audit` and `dm_unaudit`.
- All executions of a Signoff method
The event name is `dm_signoff`.
- All executions of an Adddigsignature method
The event name is `dm_adddigsignature`. By default, audit trail entries created for the `dm_adddigsignature` event are not signed by Content Server. If you want those entries signed, issue an explicit Audit method for the event and set the command line argument to require signing.
- All executions of an Addesignature method.
The event name is `dm_addesignature`. The audit trail entries for the `dm_addesignature` event are signed by Content Server automatically.
- Removal of an audit trail entry from the repository
A `dm_purgeaudit` event is generated whenever a Destroy method is executed to remove an audit trail entry from the repository or a `PURGE_AUDIT` administration method is executed. All `dm_purgeaudit` events are audited.
- User login failure

With the exception of user login failure, there are no default registry objects for these events, and you cannot turn off auditing for them. However, you can issue an explicit Audit method against these events. If you do, Content Server creates the audit trail entry based on the criteria you define in the Audit method for the event and target. Similarly,

you can issue an `Unaudit` method to remove your registration. Content Server will return to creating the default audit trail entry for the event and target.

You can turn off auditing of login failures. However, you must have `Config Audit` permission to do so. Login failures are recorded in `dm_audittrail` objects. The event name is `dm_logon_failure`.

Auditing system events

Content Server recognizes a wide range of system events. System events are associated with API methods, lifecycles, workflows, and jobs. (Refer to [Appendix B, System Events](#), in the *Content Server API Reference Manual* for a list of recognized system events.) When an audited system event occurs, Content Server automatically generates the audit trail entry.

Note: You cannot use a system event to audit the DQL `EXECUTE` statement or a DQL `INSERT` or `DELETE` statement that modifies registered tables. If you want to audit such events, you must define and audit them as application events. [Auditing application events, page 409](#), describes how that is done.

To initiate auditing of a system event, you must have `Config Audit` privileges. You can use Documentum Administrator or an `Audit` method. Instructions for using Documentum Administrator to register for auditing are found in the Documentum Administrator online help. To use `Audit`, refer to the description in the `IDfAuditTrailManager` interface in the Javadocs for the `DFC` method or to [Audit, page 115](#), in the *Content Server API Reference* for the `DMCL` method.

Initiating auditing for a system event creates a `dmi_registry` object that records the event's registration for auditing.

Remember, the more events you audit, the more audit trail entries are created in the repository. To conserve space, audit the minimum number of events necessary and manage the audit trail actively. (Refer to [Removing audit trail entries, page 430](#), for more information about audit trail management.)

Auditing application events

To audit application events, an application must recognize when the event occurs and create the audit trail entry programmatically. Application events are not recognized by Content Server.

You can use an `Audit` method to create a `dmi_registry` object for an application event. If you do, then applications can check for the presence of the registry object to determine whether or not to create the audit trail entry for the event. This is the most flexible way

to administer and manage auditing of application events. To turn auditing on or off for an application event, you simply create or destroy the event's registry object. You don't have to rewrite and recompile your application each time.

When the event occurs, use a `Createaudit` method to create the audit trail entry. If you use `Createaudit`, Content Server sets all the entry attributes except the generic attributes (`string_n` and `id_n`). If you use `Create`, `Set`, and `save` methods to create the entry, Content Server sets only the `r_gen_source` attribute automatically. The `r_gen_source` attribute indicates whether the audit trail entry was created as a result of system event or an application event. 0 indicates that the audit trail object was created by a user other than Content Server.

Signing audit trail entries

As an added security feature, audit trail entries can be signed by Content Server. Signing an entry increases security by making it possible to detect whether the entry was changed after it was saved to the repository. A signature on an audit trail entry applies to the entry and an associated `dmi_audittrail_attrs` object, if one exists.

Use the `sign_audit` argument in the `Audit` method to request that audit trail entries for a particular event be signed. This argument is `FALSE` by default. Setting it to `T` (`TRUE`) directs Content Server to sign the audit trail entries generated by that audit registration. For example, suppose you issue the following `Audit` method:

```
dmAPIExec ("audit,S0,090000021648ac2f,dm_checkin,,,,T,
'modify_date,modified_by'")
```

The example registers the document identified by the object ID for auditing whenever the document is checked in to the repository. The method requests that the audit trail entry be signed by Content Server and the values of the `modify_date` and `modified_by` attributes be recorded in the audit trail entry.

Note: You cannot set the `sign_audit` argument to `T` if the event is `"all"`, `"dm_all"`, or `"dm_all_workflow"`.

Content Server uses the AEK (Administration Encryption Key) to sign the entries. The AEK is a symmetric key that is created when Content Server is installed. To create the signature, Content Server obtains the value of the `_sign_data` computed attribute for the audit trail entry and generates a hashed value from that output. The hashed value is then encrypted using the AEK. The final result is stored in the `audit_signature` attribute of the audit trail entry.

For information about how to use a signature to verify the security of an entry, refer to [Verifying signed audit trail entries, page 429](#).

Conflicting registration resolution

Sometimes Content Server might find multiple event registrations that could apply to a single occurrence of an event. That is, the event might have multiple registry objects that could be used to generate the audit trail record. This can occur if:

- One registration registers an object type and its subtypes for an event and another registers one of the subtypes specifically for the event
- One registration registers an object type for an event and another registration registers a specific instance of the object type for the event.
- One registration registers an object type for an event and defines a controlling application and another registers the same object type for the same event but defines a policy ID.

If the conflict is between a general and a more specific registration (the first two cases listed above), Content Server uses the more specific registry object to generate the audit trail entry.

For example, suppose you register the `dm_save` event for the `dm_document` object type with the `audit_subtypes` argument set to `T` and do not identify any attributes in the `attribute_list` argument. And, suppose you have also registered the `dm_save` event on `sop_doc`, a `dm_document` subtype and have identified three attributes in the `attributes_list` argument. When an `sop_doc` document is saved, Content Server uses the registry object that identifies the `sop_doc` object type as the target of the event rather than the more general registry object that identifies `dm_document` objects as the target.

In the third situation, in which the difference in the registrations is that one has a controlling application defined and the other has a policy ID defined, Content Server uses the registration that defines a controlling application to generate the audit trail entry.

Suppose there are two registry objects for the `dm_save` event for the `dm_document` object type. One registration defines the controlling application as `dm_dcm` and does not identify any attributes for inclusion in the audit trail entry. The other defines a policy ID and defines the attribute list as `"title,keywords"`. When a save event occurs on a document controlled by `dm_dcm`, Content Server creates the audit trail entry based on the registration that includes the controlling application definition. Consequently, the audit trail entry does not include the values of the title and keywords attributes.

If you are unsure which registration was used to generate an audit trail entry, examine the `registry_id` attribute of the audit trail entry. The attribute records the object ID of the `dm_registry` object that generated the event.

Stopping auditing

To stop auditing a system event, use Documentum Administrator or an `Unaudit` method. You can also use `Unaudit` to destroy a registry object that represents an application event. Stopping auditing through Documentum Administrator or using the `Unaudit` method destroys the registry object that represents the event.

You must have `Config Audit` privileges to stop auditing by destroying the registry object for an event.

For more information about `Unaudit`, refer to the `IDfAuditTrailManager` interface in the DFC or to [Unaudit, page 466](#) in the *Content Server API Reference Manual* for the `DMCL` method.

Viewing audit trails

Use Documentum Administrator to view an audit trail. What you can view is determined by your privileges and the value in an audit trail entry's `i_audited_obj_class` attribute. The following rules are applied to entries for system events:

- Users with `Superuser` or `View Audit` privileges can view any audit trail entry.
- Users without `Superuser` or `View Audit` privileges can view audit trail entries for `SysObject`-related system events (`i_audited_obj_class` is set to 0) if they have at least `Browse` privileges for the audited object.
- Users without `Superuser` or `View Audit` privileges cannot view audit trail entries for `ACL`, `group`, and `user`-related events (`i_audited_obj_class` is set to 1, 2, or 3)
- Users who are the owner of an audited object can always view audit trail entries for the object.

All users, regardless of privilege level, can view audit trail entries for application events.

If you do not have `Superuser` or `View Audit` privileges, the following attributes of the audit trail entries are excluded from view for both system and application events:

- | | |
|----------------------------------|------------------------------|
| • <code>acl_name</code> | • <code>object_name</code> |
| • <code>acl_domain</code> | • <code>object_type</code> |
| • <code>attribute_list</code> | • <code>owner_name</code> |
| • <code>attribute_list_id</code> | • <code>session_id</code> |
| • <code>chronicle_id</code> | • <code>version_label</code> |

Querying and retrieving audit trail entries

You can query and retrieve audit trail objects directly, using a DQL SELECT statement or an API Fetch method. Which audit trail objects are returned is subject to the same restrictions as those imposed for viewing audit trail entries through Documentum Administrator, with one exception.

You must have Superuser or View Audit privileges to query or retrieve objects of type `dmi_audittrail_attrs`. If you do not have either of those privileges and wish to retrieve all audited attribute values for a particular event, including any stored in a `dmi_audittrail_attrs` object, query the computed attribute, `_attribute_list_values`.

Interpreting audit trails of API, workflow, and lifecycle events

Audit trail entries for API, workflow, and lifecycle events are stored in the repository as audit trail objects. Each audit trail object records one occurrence of a particular event. The attributes in an audit trail object describe the event.

Audit trail attributes with a common purpose

An audit trail object has many attributes that are used by all system events and are available for use in user-defined audit trail entries. The attributes that have a common purpose for entries generated by Content Server include all the attributes other than the `string_x` and `id_x` attributes. For example, the `user_id` attribute contains the `dm_user` object ID of the user who caused the event to happen. For a complete listing of these attributes, refer to [Audit Trail, page 88](#) in the *EMC Documentum Object Reference Manual*.

Attributes available for varied purposes

Each audit trail object has five ID-datatype attributes and five string attributes. These attributes are named generically, `id_1` to `id_5` and `string_1` to `string_5`. In audit trail entries for workflows and lifecycle events, Content Server uses these attributes to store information specific to the particular kinds of events. Some of the events generated by API methods other than workflow or lifecycle-related methods use these generic attributes and some do not. A user-defined event can use these attributes to store any information wanted.

Use in API audit trails

Table 11–7, page 414 describes how Content Server uses the generic string and id attributes in audit trails generated by API methods. Only those API events that use the generic attributes are listed. If an event is not listed in this table, Content Server does not use the generic attributes in audit trails generated by that event.

Table 11-7. Usage of generic attributes by API events

Event	Generic attribute use
Addsignature	<p>string_1, User name provided as an argument to the method or the name of the connected user if the user argument was not specified</p> <p>string_2, Reason for the signing</p>
Add dynamic group	None are used
Addsignature	<p>string_1, User who signed the object</p> <p>string_2, Justification text</p> <p>string_3, The number of the signature, the name of the method used to generate the signature, and the pre-signature hash argument. For example: <i>2/PDFSign/pre-signature hash_argument</i></p> <p>string_4, Hash of the primary content (page number 0)</p> <p>string_5, Hash of the signed content. Note that if the signed content is the primary content (rather than a rendition), this value is the same as the hash in string_4.</p>
Addnote	<p>id_1, Object ID of document to which the note is attached</p> <p>id_2, Object ID of the note</p>
Addrendition	<p>id_2, Object ID of the content object</p> <p>string_1, Format of the rendition</p> <p>string_2, Either “Replace Old Rendition” or “Save New Rendition”, depending on whether the added rendition replaces an existing rendition or is a new rendition.</p>

Event	Generic attribute use
Addretention	id_1 , Object ID of the retainer object
Appendcontent	See Setfile
Appendfile	See Setfile
Appendpart	id_1 , Object ID of the virtual document to which you are appending the component id_2 , Object ID of the component id_3 , Object ID of the containment object that links the virtual document and the component string_1 , The version label of the component string_2 , The use_node_ver_label setting string_3 , The follow_assembly setting string_4 , The copy_child setting
Assume	string_1 , The success or failure of the user authentication
Audit	string_1 , The audited operation
Authenticate	string_1 , The success or failure of the user authentication
Bindfile	See Setfile
Checkin	id_1 , Object ID of the version from which the new version created by the checkin was derived
Connect	string_1 , Identifies the authentication mechanism used to authenticate the user. Values are: ticket, for ticketed login trusted, for trusted login unified, for Windows unified login OS password, for OS password authentication plugin password, for plugin authentication LDAP password, for LDAP authentication OS external, for authentication with OS password by an external password checking program LDAP external, for authentication by an external password checking program that uses LDAP string_4 , Host name of the client machine from which the user connected string_5 , IP address of the client machine from which the user connected
Destroy	Destroy uses the generic attributes only when the object to be destroyed is a dm_audittrail object or a

Event	Generic attribute use
	subtype of dm_audittrail. For details, see the Purge Audit entry in this table.
Getcontent	See Getfile
Getfile	id_1 , Object ID of the content object for the content file
Getlogin	id_1 , Object ID of the user object representing the user identified in string_1. string_1 , Name of the user for whom the ticket was requested
Getpath	See Getfile
Insertcontent	See Setfile
Insertfile	See Setfile
Insertpart	id_1 , Object ID of the virtual document to which you are inserting the component id_2 , Object ID of the component id_3 , Object ID of the containment object that links the virtual document and the component string_1 , The version label of the component string_2 , The use_node_ver_label setting string_3 , The follow_assembly setting string_4 , The copy_child setting
Kill	id_1 , Session ID of the terminated session
Link	id_1 , Object ID of the folder to which the object is linked. id_2 , Object ID of the linked object string_1 , Name of the folder to which the object is linked string_2 , Name of the linked object

Event	Generic attribute use
Logon Failure	<p>string_1, User_name value</p> <p>string_2, User name as entered by the user</p> <p>Note: string_1 is empty if the user enters an invalid user name. If the user enters a valid user name, string_1 and string_2 are the same value.</p>
Mark	<p>string_1, Name of the version label</p> <p>Note: One audit trail entry is created for each label assigned in the Mark method.</p>
Move Content	<p>string_1, Name of the storage area from which the content was moved</p> <p>string_2, Name of the storage area to which the content was moved.</p> <p>id_1, Object ID of the SysObject containing the moved content file.</p>
Purge Audit	<p>string_1, the time_stamp value of the first deleted audit trail entry in the transaction</p> <p>string_2, the time_stamp value of the last deleted audit trail entry in the transaction</p> <p>string_3, the actual number of audit trail entries deleted in the transaction</p> <p>string_5, the list of arguments defined in the method command line</p> <p>id_1, the object ID of the first audit trail entry deleted by the transaction</p> <p>id_2, the object ID of the last audit trail entry deleted by the transaction</p>
Removecontent	<p>id_1, Object ID of the content object representing the content file removed from the SysObject.</p> <p>string_1, The page that was removed.</p>
Remove dynamic group	None are used

Event	Generic attribute use
Removenote	id_1 , Object ID of the object to which the note was attached
Removepart	id_2 , Object ID of the note id_1 , Object ID of the virtual document from which you are removing the component id_2 , Object ID of the component id_3 , Object ID of the containment object that links the virtual document and the component string_1 , If specified, the order_no that identifies the component to be removed
Removerendition	id_2 , Object ID of the content object representing the rendition's content file string_1 , Rendition format
Removeretention	id_1 , Object ID of the retainer object
Setcontent	See Setfile
Setfile	id_1 , Object ID of the content object for the content file string_1 , Name of the API string_2 , Name of the file (unused for Appendcontent, Bindfile, Inserttcontent, Setcontent) string_3 , Page number of the content file string_4 , Format of the content file
Setpath	See Setfile
Setoutput	id_1 , Object ID of the workflow id_2 , Object ID of the work item string_1 , Activity sequence number string_2 , Activity name string_5 , Output port name

Event	Generic attribute use
Setretentionstatus	<p>string_1, Original status of the retainer</p> <p>string_2, New status of the retainer</p>
Unaudit	string_1 , Name of the operation from which auditing was removed
Unlink	<p>id_1, Object ID of the folder to which the object is linked.</p> <p>id_2, Object ID of the linked object</p> <p>string_1, Name of the folder to which the object is linked</p> <p>string_2, Name of the linked object</p>
Unmark	string_1 , Name of the version label.
Updatepart	<p>id_1, Object ID of the virtual document into which you are inserting the component</p> <p>id_2, Object ID of the component</p> <p>id_3, Object ID of the containment object that links the virtual document and the component</p> <p>string_1, The version label of the component</p> <p>string_2, The use_node_ver_label setting</p> <p>string_3, The follow_assembly setting</p> <p>string_4, The copy_child setting, if specified</p> <p>string_5, The order_no if specified</p>

Use in lifecycle audit trails

Table 11–8, page 420 describes how Content Server uses the generic string and id attributes in audit trails generated by lifecycle events.

Table 11-8. Usage of generic attributes by lifecycle events

Event	Generic attribute use
Attach	id_1 , Object ID of the business policy string_1 , State to which object is being attached
Demote	id_1 , Object ID of the business policy string_1 , State from which object was demoted string_2 , State to which the object was demoted
Install	Does not use the generic attributes.
Promote	id_1 , Object ID of the business policy string_1 , State from which object was promoted string_2 , State to which object was promoted
Resume	id_1 , Object ID of the business policy string_1 , State to which the object is returned
Suspend	id_1 , Object ID of the business policy string_1 , The object's business policy state at the time of suspension
Uninstall	Does not use the generic attributes.
Validate	string_1 , Number of states in the business policy.

Use in workflow audit trails

[Table 11-9, page 420](#) describes how Content Server uses the generic string and id attributes in audit trails generated by workflow events.

Table 11-9. Usage of generic attributes by workflow events

Event	Generic attribute use
Abort workflow	id_1 , Object ID of the workflow

Event	Generic attribute use
Add attachment	<p>id_1, Object ID of the workflow</p> <p>id_5, Object ID of the attached object</p> <p>string_5, Name of the attached object</p>
Add note	<p>id_1, Object ID of the workflow</p> <p>id_5, Object ID of the note object</p> <p>string_1, Activity sequence number</p> <p>string_2, Activity name</p> <p>string_5, Value of the note's keep_permanent flag</p>
Add package	<p>id_1, Object ID of the workflow</p> <p>id_2, Object ID of the work item (used only if Addpackage is called with the work item referenced in the arguments)</p> <p>id_5, Object ID of the document in the package. If there are multiple documents, there are a corresponding number of audit trail entries, each with a different value in id_5.</p> <p>string_1, Activity sequence number</p> <p>string_2, Activity name</p> <p>string_3, Package component names identified in the component_name argument. This is only set if names are specified in the Addpackage method and package control is not enabled.</p> <p>string_5, Package name</p>
Auto Delegation of Activity Failed	<p>id_1, Object ID of the workflow</p> <p>string_1, Activity sequence number</p> <p>string_2, Activity name</p>

Event	Generic attribute use
Auto Transition of Activity	<p>id_1, Object ID of the workflow</p> <p>string_1, Activity sequence number</p> <p>string_2, Activity name</p> <p>string_5, Index position of the TRUE condition in the <code>dm_cond_expr</code> object examined for the transition.</p> <p>(This event is supported for backwards compatibility. The Portselect event provides additional or more current information about an automatic transition.)</p>
Change activity instance state	<p>id_1, Object ID of the workflow</p> <p>id_2, Object ID of the work item</p> <p>id_5, Value in the <code>r_exec_results</code> attribute of the work item</p> <p>string_1, Activity sequence number</p> <p>string_2, Activity name</p> <p>string_5, Value of the <code>return_value</code> attribute of the work item</p>
Change process state	<p>string_3, Previous state</p> <p>string_4, New state</p>
Change supervisor	<p>id_1, Object ID of the workflow</p> <p>string_4, New supervisor name</p> <p>string_5, Old supervisor name</p>
Change workflow state	<p>id_1, Object ID of the workflow</p> <p>string_3, Previous state</p> <p>string_4, New state</p>

Event	Generic attribute use
Change work item priority	<p>id_1, Object ID of the workflow that contains the work item</p> <p>id_2, Object ID of the work item</p> <p>string_1, Sequence number of the activity that generated the work item</p> <p>string_2, Name of the activity</p> <p>string_3, Old priority value</p> <p>string_5, New priority value</p>
Completed work item	<p>id_1, Object ID of the workflow</p> <p>id_2, Object ID of the work item</p> <p>id_5, For automatic work items: Object ID of the document containing the results of the execution. (This is the value in the <code>r_exec_results</code> attribute.)</p> <p>string_1, Activity sequence number</p> <p>string_2, Activity name</p> <p>string_3, Comma-separated list of work item attributes and their values. The attributes are: <code>user_time</code>, <code>user_cost</code>, <code>a_wq_name</code>, <code>a_wq_flag</code>, and <code>a_wq_doc_profile</code>. (<code>a_wq_name</code> and <code>a_wq_doc_profile</code> are enclosed in double quotes)</p> <p>string_5, Value in the <code>return_value</code> attribute of work item</p>
Create workflow	<p>id_1, Object ID of the workflow</p> <p>string_4, Supervisor name</p> <p>string_5, Name of the workflow</p>

Event	Generic attribute use
Delegated work item	id_1 , Object ID of the workflow id_2 , Object ID of the work item string_1 , Activity sequence number string_2 , Activity name string_3 , Activity type (Manual or Automatic)
Finish workflow	id_1 , Object ID of the workflow
Install workflow or activity definition	Does not use the generic attributes.
Invalidate workflow or activity definition	Does not use the generic attributes.
Pause work item	id_1 , Object ID of the workflow id_2 , Object ID of the work item string_1 , Activity sequence number string_2 , Activity name
Port select	id_1 , Object ID of the workflow string_1 , Activity sequence number string_2 , Activity name string_5 , Selected output port. If multiple ports are selected, a corresponding number of audit trail entries are created, each with a different value in string_5.
Pseudo-completed work item	id_1 , Object ID of the workflow id_2 , Object ID of the work item string_1 , Sequence number of the activity string_2 , Name of the activity string_3 , State of the work item prior to pseudo-completion string_5 , Name of the task owner

Event	Generic attribute use
Remove attachment	<p>id_1, Object ID of the workflow</p> <p>id_5, Object ID of the attached object that was removed</p> <p>string_5, Name of the attached object that was removed</p>
Remove note	<p>id_1, Object ID of the workflow</p> <p>id_5, Object ID of the note object. If the remove note event is triggered by a remove package event that removes a package with multiple notes attached to its components, there are multiple remove note audit trail entries, each with a different id_5 value.</p> <p>string_1, Activity sequence number</p> <p>string_2, Activity name</p>
Remove package	<p>id_1, Object ID of the workflow</p> <p>id_5, Object ID of the document in the package. If there are multiple documents, there are a corresponding number of audit trail entries, each with a different value in id_5.</p> <p>string_1, Activity sequence number</p> <p>string_2, Activity name</p> <p>string_5, Package name</p>
Repeat work item	<p>id_1, Object ID of the workflow</p> <p>id_2, Object ID of the work item</p> <p>string_1, Activity sequence number</p> <p>string_2, Activity name</p>

Event	Generic attribute use
Resume work item	<p>id_1, Object ID of the workflow</p> <p>id_2, Object ID of the work item</p> <p>string_1, Activity sequence number</p> <p>string_2, Activity name</p>
Selected work item (a work item has been acquired)	<p>id_1, Object ID of the workflow</p> <p>id_2, Object ID of the work item</p> <p>string_1, Activity sequence number</p> <p>string_2, Activity name</p>
Sign off work item	<p>id_1, Object ID of the workflow</p> <p>id_2, Object ID of the work item</p> <p>id_5, Object ID of the document in the package. If there are multiple documents, there are a corresponding number of audit trail entries, each with a different value in id_5.</p> <p>string_1, For Windows, the user's domain and user name (used to validate signature)</p> <p>string_5, Text provided by <i>reason</i> argument in Signoff method.</p>
Started work item	<p>id_1, Object ID of the workflow</p> <p>id_2, Object ID of the work item</p> <p>string_1, Activity sequence number</p> <p>string_2, Activity name</p> <p>string_3, Comma-separated list of work item attributes and their values. The attributes are: <i>r_priority</i>, <i>a_wq_name</i>, and <i>a_wq_doc_profile</i>. <i>a_wq_name</i> and <i>a_wq_doc_profile</i> are enclosed in double quotes.</p> <p>string_5, Value in the <i>dependency_type</i> attribute of the corresponding queue item object.</p>

Start workflow	<code>id_1</code> , Object ID of the workflow
Uninstall workflow or activity definition	Does not use the generic attributes.
Validate workflow or activity definition	Does not use the generic attributes.

Interpreting ACL and group audit trails

Audit trail entries generated when ACLs are created, changed, or destroyed are recorded in `dm_audittrail_acl` objects. Entries generated when groups are created, changed, or destroyed are recorded in `dm_audittrail_group` objects. The `dm_audittrail_acl` and `dm_audittrail_group` object types are subtypes of the `dm_audittrail` type.

The attributes defined for the `dm_audittrail_acl` object type store information about an audited ACL. The attributes identify the event (`dm_save`, `dm_saveasnew`, or `dm_destroy`), the target ACL, the ACL entries or changes to the entries. For example, suppose you create an ACL with the following attribute values:

```
r_object_id:451e9a8b0001900
r_accessor_name      [0]:dm_world
                    [1]:dm_owner
                    [2]:John Doe
r_accessor_permit    [0]:6
                    [1]:7
                    [2]:3
r_accessor_xpermit   [0]:0
                    [1]:0
                    [2]:196608
r_is_group           [0]:F
                    [1]:F
                    [2]:F
```

The audit trail acl object created in response has the following attribute values:

```
r_object_id:<audit trail acl obj ID>
audited_obj_id      :451e9a8b0001900
event_name          :dm_save
string_1            :Create
accessor_operation[0] :I
                    [1] :I
                    [2] :I
accessor_name       [0] :dm_world
                    [1] :dm_owner
                    [2] :John Doe
accessor_permit     [0] :6
                    [1] :7
                    [2] :3
accessor_xpermit    [0] :0
                    [1] :0
                    [2] :196608
```

```
application_permit [0]:
                  [1]:
                  [2]:
permit_type       [0]:0
                  [1]:0
                  [2]:0
is_group          [0] :F
                  [1] :F
                  [2] :F
```

The `event_name` records the repository event, a `Save` method, that caused the creation of the audit trail entry. The `string_1` attribute records the event description, in this case, `Create`, indicating the creation of an ACL. The `accessor_operation` attribute describes what operation was performed on each accessor identified at the corresponding index position in `accessor_name`. The `accessor_permit` and `accessor_xpermit` attributes record the permissions assigned to the user (or group) identified in the corresponding positions in `accessor_name`. Finally, the `is_group` attribute identifies whether the value in `accessor_name` at the corresponding position represents an individual user or a group.

Now suppose you change the ACL. The changes you make are:

- Add the Sales group
- Remove John Doe
- Change the world's access to None

The resulting audit trail acl object has the following attributes:

```
r_object_id       :<audit trail acl obj ID>
audited_obj_id    :451e9a8b0001900
event_name        :dm_save
string_1          :Save
accessor_operation [0] :U
                  [1] :I
                  [2] :D
accessor_name     [0] :dm_world
                  [1] :Sales
                  [2] :JohnDoe
accessor_permit   [0] :0
                  [1] :2
                  [2] :3
accessor_xpermit  [0] :0
                  [1] :0
                  [2] :196608
application_permit [0] :
                  [1] :
                  [2] :
permit_type       [0] :0
                  [1] :0
                  [2] :0
is_group          [0] :F
                  [1] :T
                  [2] :F
```

`dm_world` is found in `accessor_name[0]`. Consequently, the values in the corresponding position in the `accessor_operation`, `accessor_permit`, and `accessor_xpermit` attributes

identify the changes made to `dm_world`. In this case, the operation is U, meaning update. The values in `accessor_permit` and `accessor_xpermit` show the updated permissions for `dm_world`.

Sales is found in `accessor_name[1]`. The value in `accessor_operation[1]`, I, shows that an entry for Sales was added (inserted) into the ACL. The values in `accessor_permit` and `accessor_xpermit` show the permissions assigned to Sales.

JohnDoe is found in `accessor_name[2]`. The value in `accessor_operation[2]`, D, indicates that the entry for JohnDoe was removed from the ACL. The values in the corresponding positions in `accessor_permit` and `accessor_xpermit` identify the permissions previously assigned to JohnDoe.

Audit trail group objects are interpreted similarly. The values in the corresponding index positions in `users_names` and `users_names_operations` represent one individual user who is a member of the audited group. The values in the corresponding positions in `groups_names` and `groups_names_operation` represent one group that is a member of the audited group. The operations attribute defines what operation was performed on the member at the corresponding names attribute.

Verifying signed audit trail entries

A signed audit trail entry is an entry that has a value in the `audit_signature` attribute. The value is a signature generated by Content Server when the entry was created. A signature on an entry is an added security feature because it allows you to determine whether the entry was changed after it was saved to the repository.

If you need to check whether an entry was changed, use the `Verifyaudit` method. This method regenerates the signature and compares the newly generated signature with the signature stored in the `audit_signature` attribute. If the method returns T, then the audit trail entry is secure and has not been changed.

If you are working in a repository that has multiple servers, the method fails if the servers are using different AEKs and a server attempts to verify a signature created by another server with a different AEK. To ensure that this does not occur, either all servers for a particular repository must use the same AEK or the application must check that the server issuing the `Verifyaudit` method is the same server that generated the signature. The server that generates the signature is recorded in the `host_name` attribute of the audit trail entry.

Removing audit trail entries

If the audit trail becomes too large, it can fill the available space in the underlying RDBMS database. If there is no space in the database, you will not be able to save any objects in your repository. Monitor the size of the audit trail carefully.

When necessary, archive audit data that you want to keep by copying it or moving it out of the audit trail. Then, run the Audit Management tool or execute the `PURGE_AUDIT` administration method to remove unneeded audit trail entries from the database. The account under which the tool runs must have Purge Audit privileges. Similarly, if you use `PURGE_AUDIT`, the user executing the method must have Purge Audit privileges.

The Audit Management tool removes only system-generated audit trail entries by default. You can reset a default parameter to direct it to remove both system- and user-generated entries or only user-generated entries. For more information on the Audit Management tool, refer to [Audit Management](#), page 468. The Audit Management tool is installed in the inactive state.

`PURGE_AUDIT` provides several options for determining which audit trail entries to remove. For example, you can remove all entries created within specified dates or all entries that reference a specific object as the target of the audit. For complete details about using this administration method, refer to [PURGE_AUDIT](#), page 275, in the *Content Server DQL Reference Manual*.

Removing an audit trail entry also removes the `dmi_audittrail_attr` object associated with the entry if one exists.

Auditing in a distributed environment

In a distributed environment, the audit trail is generated at the source repository. For example, suppose you are auditing the check out operations on a document. If a user in a remote repository checks out the document, the audit trail entry is written to the audit trail in the document's source repository.

If you are auditing a virtual document that has component documents stored in multiple repositories, the Assemble audit trail is written to the repository to which you are connected when the Assemble method is executed, but the Checkin audit trail for individual components is written to the corresponding source repository.

To collect all audit trail information for one compound document, you need to check multiple audit logs. Use the timestamp to merge the audit logs when you generate a report.

Implementing signature support

This section describes the administration tasks that support the use of an electronic signature feature in applications. There are three options for electronic signatures:

- Electronic signatures using Addesignature

Electronic signatures are implemented through Content Server and provide rigorous accountability. This option requires a Trusted Content Services license. For information on using and customizing electronic signature support, refer to [Customizing electronic signatures, page 431](#). Electronic signatures are not supported on the Linux and HP Itanium platforms.

- Digital signatures using Adddigsignature

Digital signatures are implemented in client applications, using third-party software. No additional Content Server license is needed. For instructions on supporting digital signatures, refer to [Supporting digital signatures, page 440](#).

- Simple signoffs

Simple signoffs are the least rigorous way to enforce an electronic signature requirement. No additional Content Server license is needed. For instructions on using and customizing simple signoffs, refer to [Customizing simple signoffs, page 441](#).

Customizing electronic signatures

There a variety of options for customizing electronic signatures. If you are using the default signature page template and signature creation method, you can:

- Add or remove properties from the template page
- Change the property delimiters on the page
- Change the look of the template by adding, removing, or rearranging elements on the page or changing the font and point size of the properties
- Define separate templates for different document types
- Configure the number of signatures allowed on a version of a document and whether the signature page is added to the front or end of the content.

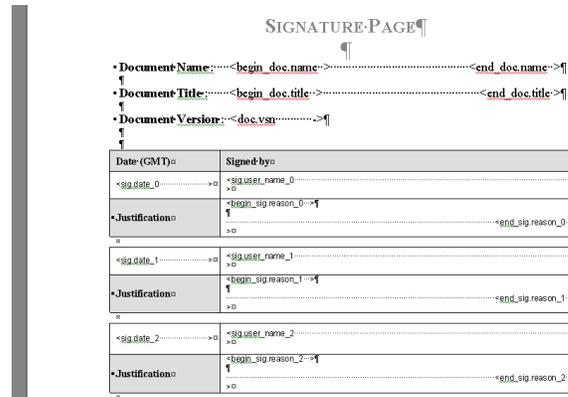
For instructions on customizing the default functionality, refer to [Customizing the default functionality, page 432](#).

If you want the signature in non-PDF format, you must implement a custom signature creation method. If you use a custom method, you can use a custom signature page template if you wish, but it is not required. The signature can be in any form that your method implements. [Creating custom signature creation methods and associated](#)

signature page templates, page 438, contains instructions for implementing a custom method.

Customizing the default functionality

Use the procedures in this section to customize the default signature page provided with Content Server. , shows a portion of the default signature page template.



The text in angle brackets (<>) identifies properties. When a signature page is created, the creation method replaces the text and angle brackets with the value of the property. For example, <begin.doc_name>...<end.doc_name> is replaced with the value of the object_name attribute, and <doc.vsn> is replaced with the object's numerical version label. Some of the properties are values of attributes for the object. Others, such as <sig.date>, are values associated with the electronic signature.

Content Server supports a set of system-defined properties for use on the default template page. The pre-defined properties represent attributes of the document being signed and values associated with the signature. You can also add user-defined properties to the signature page template.

On the template page, properties are formatted as either simple properties or textbox properties. A simple property cannot break across lines in the signature page and is represented by a single placeholder in the template. For example, <doc.vsn> on the default template is a simple property. Textbox properties represent values that may require multiple lines on the signature page. They are represented by begin and end placeholders. For example, <begin.doc.name>...<end.doc.name> represent the textbox property doc.name on the default template.

Some of the system-defined document properties can be formatted as either simple or textbox properties on a signature template page, and some can only be formatted as simple properties. All of the properties associated with the signature can be formatted as either simple or textbox properties.

Table 11–10, page 433, lists the system-defined properties associated with documents, and Table 11–11, page 434, lists those associated with signatures. In addition, you can add custom properties. For instructions, refer to [Adding or removing properties on the page](#), page 434.

Table 11-10. System-defined document properties for the signature page template

Property	Simple/textbox	Replaced with
doc.id	Simple	Object ID of the object being signed
doc.name	Simple or Textbox	Object name of the object being signed
doc.title	Simple or Textbox	Title of the object being signed
doc.path	Simple or Textbox	Folder path of the object being signed, including the object's name. For example: MyCabinet/Proposals/Engineering/NewProject.doc
doc.vsn	Simple	The numeric version label of the object being signed
doc.status	Simple	The a_status value of the object being signed
doc.owner	Simple	Name of the owner of the object being signed
doc.modifier	Simple	Name of the last person to have modified the object being signed (the value of the r_modifier attribute)
doc.modify_date	Simple	Date on which the object was last modified (the r_modify_date value)
doc.authors	Simple or Textbox	Authors of the object being signed
fdr.path	Simple or Textbox	Folder path of the object in the repository, without the document name. For example: MyCabinet/Proposals/Engineering
sig.requestor	Simple or Textbox	The name of the user requesting the current signature
sig.count	Simple	The number of signatures currently recorded for the document

Table 11-11. System-defined signature properties for the signature page template

Property	Replaced with
sig.requestor	The user issuing the Addesignature method
sig.no	Signature number
sig.user_id	User authentication name provided in the Addesignature arguments
sig.user_name	Repository user name of the user identified in the Addesignature arguments
sig.reason	The justification text for the signature
sig.date	Date and time at which the signature was generated. The format is mm/dd/yyyy hh:mm:ss
sig.utc_date	Date and time at which the signature was generated, expressed as UTC time.

Adding or removing properties on the page

You can add any of the system-defined properties or user-defined properties. When you add a system-defined property, the default signature creation method will automatically replace the property placeholder on the page with the appropriate value. If you add a user-defined property, applications must pass the property name and value to the signature creation method through the `app_properties` argument in the Addesignature method.

To add or remove a property on the default signature page template:

1. Check out the template from the repository.
The template is stored in `/Integrations/Esignatures/Templates` and its object name is `Default Signature Page Template`.
2. Open the document for editing.
3. To add a property, enter the property name enclosed in the appropriate delimiters at the location desired in the document.
By default, the delimiters are angle brackets. If you have changed that default, be sure to use the delimiters you have chosen. Note that you may also add display labels or place the value in a table or format the alignment of the property's value. (For instructions on formatting alignment, refer to [Configuring the appearance of the page, page 435](#).)
4. To remove a property, delete the entry and any associated formatting (such as label text).

5. Check in the template.
6. Use PDFWriter to generate a PDF rendition of the template.
7. Replace the current PDF rendition of the template with the new template, setting the `page_modifier` of the PDF rendition to `dm_sig_template`.
Refer to [Addrendition, page 66](#), in the *Content Server API Reference* for instructions on adding renditions.

Changing the property delimiters

On the default signature page template, the delimiters for properties on the page are angle brackets (< >). To change that default, set the `begin_tag` and `end_tag` attributes of the template's `dm_esign_template` object. You cannot set the beginning and ending delimiters to the same character. For example, if you set `begin_tag` to the pound sign (#), you cannot set `end_tag` to a pound sign. You must choose a character other than the pound sign for the ending delimiter.

To set the delimiters for properties on the default signature page template:

1. Fetch or checkout the template from the repository.
The template is stored in `/Integrations/Esignatures/Templates` and its object name is `Default Signature Page Template`.
2. Set the `begin_tag` attribute to the desired delimiter for the beginning of the property placeholders.
3. Set the `end_tag` attribute to the desired delimiter for the ending of the property placeholders.
4. Save or checkin the template.
5. Use PDFWriter to generate a new PDF rendition of the template.
6. Replace the current PDF rendition of the template with the new rendition, setting the `page_modifier` attribute of the rendition to `dm_sig_template`.
Refer to [Addrendition, page 66](#), in the *Content Server API Reference Manual* for instructions on adding renditions.

Configuring the appearance of the page

You can modify the appearance and arrangement of the default signature page in any manner allowed by a Word editor. You can add display labels, graphics, tables, table titles, and so forth. You can also change the font and point size of the properties that are placed on the page. To do that, select the property name in the editor and reset the font,

point size, or both in the editor. When the template is used, text on the resulting signature page will use the font and point size of its associated property name on the template.

You can also change the alignment of the property values on the page. You do this by positioning the property name inside the angle brackets using spaces. For example, the following property designation right-justifies the property value, with four leading spaces (each caret represents one space for the purposes of illustration):

```
<^^^^doc.name>
```

If you wanted to center the value, insert equal numbers of spaces before and after the property name:

```
<^^^^doc.name^^^^>
```

Simple property designations must include enough space to hold the longest value expected for that property. If a value is too long for the space provided, the value is truncated. The space provided is determined by counting the characters in the property designation. For example, suppose you add the user-defined property <doc_creator> to the template page, to add the name of the document's creator to the page. This property designation has 13 characters, and therefore user names longer than 13 characters would be truncated. To allow for longer document names, include spaces (shown as carets for purposes of illustration) to represent the extra characters:

```
<doc_creator^^^^^^^^>  
or  
<^^^^^^^^doc_creator>  
or  
<^^^^doc_creator^^^^>
```

To modify the appearance of the signature template page:

1. Check out the template.
The template is stored in /Integrations/Esignatures/Templates and its object name is Default Signature Page Template.
2. Open the template in a Word editor.
3. Make the desired modifications.
4. Check in the template.
5. Generate a new PDF rendition of the template using PDFWriter.
6. Replace current PDF rendition of the template with the new rendition, setting the page_modifier attribute to dm_sig_template.

Defining separate templates for different document types

The default signature page template is defined for use on any dm_document object or dm_document subtype. However, your business requirements for signature pages may

be different for different document subtypes. For example, a signed legal document might require display labels of “Plaintiff signature” or “Defendant signature” on the signature, while a medical document might require a display label of “Patient signature” on the signer’s name. You can create different default signature page templates for different object types.

Which object types a particular signature page template is applied to is defined by the `document_type` attribute of the template. A signature page template is used to generate signature pages for the object type (and its subtypes) defined in this attribute. For example, the attribute is set to `dm_document` in the default template, which means that the template can be used to generate signature pages for any `dm_document` object or any object that is a subtype of `dm_document`.

To create a default template page for a particular object type:

1. Check out the template.
The template is stored in `/Integrations/Esignatures/Templates` and its object name is `Default Signature Page Template`.
2. Open the template in a Word editor.
3. Save the template as a new document with a name appropriate for the intended use.
For example, if the new template will be used with legal documents, you might set the name to `Legal_Doc Signature Page Template`.
4. On the new copy of the source template, make the changes you need.
5. Save the new template.
6. Import or check in the new template to the repository.
The template must be defined as an object of type `dm_esign_template`. If you need help importing or checking in the new document, refer to the client documentation or online help.
7. Set the new template’s `document_type` attribute to the name of the document subtype that the template will be used to sign.
For example, if you want the template to be used for legal documents of subtype `legal_document`, set `document_type` to `legal_document`.
8. Generate a PDF rendition of the new template using PDFWriter.
9. Append the PDF rendition to new copy of the template.

Configuring the number of allowed signatures and signature positioning

The default signature page template allows up to six signatures on a document version. Also by default, the signature page is appended to the end of the content being signed. You can increase or decrease the number of allowed signatures and you can direct the

method to add the signature page to the beginning of the content. Both of these defaults are controlled by attributes of the template object.

The number of signatures allowed is controlled by the `max_signatures` attribute. The default signature page template includes six entries for signatures. If you increase the maximum number of signatures, you should also modify the template page to add entries for the additional signatures. (If you do not add the entries for the additional signatures, when those additional signatures are added to the document, the signature operation succeeds and audit trail entries are created for the event, but the signatures won't appear on the signed content.) If you decrease the maximum number, it is not necessary to modify the template page unless you want to remove the unnecessary signature entries. If you do modify the signature page template, you must generate a new PDF version of the template and replace the old PDF rendition with the new rendition.

Whether the signature page is prepended or appended to the content is controlled by the `append_to_body` attribute of the template object. By default, this attribute is T (TRUE), meaning to append the signature page to the end of the content. Setting this attribute to F (FALSE) causes the signature creation method to prepend the signature page to the beginning of the content.

You can set either attribute using the API or DQL.

It is not necessary to generate a new PDF of the template when you change the template's positioning.

Creating custom signature creation methods and associated signature page templates

This section outlines the basic procedure for creating and installing a user-defined signature creation method. Documentum provides standard technical support for the default signature creation method and signature page template installed with the Content Server software. For assistance in creating, implementing, or debugging a user-defined signature creation method or signature page template, contact Documentum Professional Services or Documentum Developer Support.

If you do not want to use the default functionality, you must write a signature creation method. Using a signature page template is optional if you write a custom program.

Creating custom signature-creation methods

Use the following guidelines when writing the method:

- The method should check the number of signatures currently on the object to ensure that adding another signature does not exceed the maximum number of signatures for the document.

- The method must return 1 if it completes successfully or a number greater than 1 if it fails. The method cannot return 0.
- If the trace parameter is passed to the method as T (TRUE), the method should write trace information to standard output.

The Addesignature method passes the parameters listed in [Table 11-12, page 439](#) to the method.

Table 11-12. Parameters passed by Addesignature to the signature-creation method

Parameter	Description
docbase	Name of the repository
user	Name of the user who is signing
doc_id	Object ID of the document to be signed
file_path	The name and location of the content file to be signed in the file system.
signature_properties	<p>A set of property and value pairs that contain data about the current and previous signatures. The information can be used to fill in a signature page. The set includes:</p> <ul style="list-style-type: none"> • sig.requester_0...<i>n</i> • sig.no_0...<i>n</i> • sig.user_id_0...<i>n</i> • sig.user_name_0...<i>n</i> • sig.reason_0...<i>n</i> • sig.date_0...<i>n</i> • sig.utc_date_0...<i>n</i> <p>The number at the end of each parameter corresponds to the number of the signature to which the information applies.</p>
application_properties	User-defined set of attribute names and values specified in the Addesignature command line.
trace	If tracing is turned on for SysObjects, this parameter is passed as T (TRUE).
passThroughArgument1	User-defined information specified in the Addesignature command line.
passThroughArgument2	User-defined information specified in the Addesignature command line.

Use the following procedure to create a custom method.

To create a custom signature-creation method:

1. Write the method program.
2. Create a dm_method object that references the method program.

[Implementing a method, page 134](#), describes how to create method objects.

To use the custom method, applications must specify the name of the method object that represents the custom method in the Addesignature arguments.

Creating custom signature page templates

If you create a new signature template page, you define the format, content, and appearance of the page. You can store the template as primary content of an object or as a rendition. For example, you might create an XML source file for your template and generate an HTML rendition for use by your custom signature creation method. If you store the template as a rendition, set the template's page modifier to dm_sig_template. Setting the page modifier to dm_sig_template ensures that the Rendition Manager administration tool will not remove the template rendition.

Tracing electronic signature operations

You can trace the operations of the Addesignature method and the default signature creation method by setting the trace level for the DM_SYSOBJECT facility to 5 (or higher) using the Trace method:

```
trace, c, 5, , DM_SYSOBJECT
```

The tracing information for the Addesignature and Verifiesignature methods is recorded in the session log file. The tracing information for the signature creation method is recorded in the server log file.

If you are using a custom signature creation method, trace messages generated by the method written to standard out are recorded in the server log file if tracing is enabled.

Note: When tracing is enabled, the Addesignature method passes the trace parameter set to T (TRUE) to the signature creation method.

Supporting digital signatures

Digital signatures, like electronic signoffs, are a way to enforce accountability. Digital signatures are obtained using third-party client software and embedded in the content. The signed content is then stored as primary content or a rendition of the document. For

example, you might choose to implement digital signing using based on Microsoft Office XP, in which case the signature is typically embedded in the content file and stored in the repository as primary content for the document.

The implementation and management of digital signatures is almost completely within the context of the client application. (For a description of how applications implement use of digital signatures, refer to [Digital signatures, page 101](#), in *Content Server Fundamentals*.) The only system administration task is registering the `dm_adddigsignature` event for signature by Content Server. This is an optional task. When an application issues the `Adddigsignature` method, to record a digital signoff in an audit trail entry, that entry can itself be signed by the Content Server. To configure signing by the server, an explicit `Audit` method must be issued for the `dm_adddigsignature` event. For information about how Content Server signatures on audit entries are implemented and how to configure their use, refer to [Signing audit trail entries, page 410](#).

Customizing simple signoffs

Simple signoffs are implemented using a `Signoff` method and a signature validation program. Documentum provides a default signature validation program that authenticates a user based on the user's user authentication name and password passed as arguments in the `Signoff` method. If the authentication succeeds, Content Server creates an audit trail entry of event type `dm_signoff` that records what was signed, who signed it, and some information about the context of the signing. The audit trail entry is not signed by Content Server.

You can customize the simple signoff feature by:

- Creating an alternate validation program that uses the user authentication name and password to validate the user
- Passing data other than a password through `Signoff`'s password argument and creating a custom validation method to authenticate the user with that data

For example, you might pass some biometric data and then validate that using a custom validation program.

- Execute an `Audit` method to force Content Server to the sign the `dm_signoff` events or to add additional information to the entries

Documentum provides standard technical support for the default signature validation method installed with the Content Server software. For assistance in creating, implementing, or debugging a customized signature validation program or a user-defined signature validation method, contact Documentum Professional Services or Documentum Developer Support.

Customizing the signature validation program

Use the following procedure to create and implement a customized simple signoff.

To use a custom signature validation program:

1. Create a custom signature validation program.
The program must accept two arguments: the user's name and the signature data passed using the `user_password` argument of the Signoff method. If the personal data is binary, you can use the `uuencode` program to convert the data to a string. Data passed in the `user_password` argument cannot be longer than 127 bytes.
The validation program must return 0 if it succeeds; otherwise, it must return 1.
2. Create a location object that identifies where the program is installed.
3. Modify the `signature_chk_loc` attribute in the server config object to point to the location object created in Step 2.
The `signature_chk_loc` attribute identifies the location of the signature validation program to Content Server.

Registering for notification

Users can register the `dm_signoff` event so that when the Signoff method is executed, the server sends mail notifications to users. You do not need to register a separate audit event, because the server always generates an audit trail for Signoff executions.

Querying the audit trail for signoffs

To return a collection of document objects signed by a specific user within a certain time frame, use a DQL statement like the following:

```
SELECT "audited_obj_id" FROM "dm_audittrail" WHERE
"event_name" = 'dm_signoff' AND
"user_name" = 'tom' AND
substr ("audited_obj_id", 1, 2) = '09'AND
"time_stamp" >= DATE('01/01/1998', 'dd/mm/yy') AND
"time_stamp" <= DATE(TODAY)
```

To find everyone who signed off a specific object whose ID is xxx, use the following DQL statement:

```
SELECT "user_name" FROM "dm_audittrail" WHERE
"audit_obj_id" = 'xxx' AND
"event_name" = 'dm_signoff'
```

Managing the encryption keys

In each Content Server software installation there is one master key, called the Administration Encryption key, or AEK. Additionally, in each repository, there is one repository encryption key.

The AEK

There is one AEK in each Content Server software installation. It is created and installed during the Content Server software installation procedure or when an existing repository is upgraded from a version prior to 5.2. The AEK is used to encrypt:

- The repository keys
- Documentum passwords, such as those used by Content Server to connect to the RDBMS, an LDAP directory server, or other repositories

The AEK is itself encrypted using a default passphrase provided by Documentum. You can change the passphrase to a custom passphrase using a utility provided by Documentum. (Refer to [Changing a passphrase, page 447](#), for instructions.) Using a custom passphrase is recommended.

The AEK is installed in the following location:

On Windows:

```
%DOCUMENTUM%\dba\secure\aek.key
```

On UNIX:

```
$DOCUMENTUM/dba/secure/aek.key
```

The file is a read only file. You cannot change the AEK file.

Content Server and all server processes and jobs that require access to the AEK use the following algorithm to find it:

1. If a location is explicitly passed, look for the AEK in that location.
2. If the AEK is not found in the specified location or a location is not passed, use the location defined in the DM_CRYPTO_FILE environment variable.
3. If the DM_CRYPTO_FILE environment variable is not available, assume that the location is %DOCUMENTUM%\dba\secure\aek.key (\$DOCUMENTUM/dba/secure/aek.key).

Sharing the AEK or passphrase

If there are multiple Documentum products installed on one host, the products can use different AEKs or the same AEK.

If the products use different AEKs, they can use the same passphrase. If you want that passphrase to be a custom passphrase, perform the following procedure after you install the Content Server software.

To implement the same passphrase for multiple products:

1. Shutdown Content Server if it is running.
2. Run the `dm_crypto_change_password` to change the passphrase to a custom passphrase.
Refer to [Changing a passphrase, page 447](#) for instructions.
3. Run the `dm_crypto_boot` utility using the `-all` argument.
Refer to [Using dm_crypto_boot, page 446](#) for instructions.
4. Restart Content Server.
5. Install the remaining products on the host machine.

If the products use one AEK, they must use the same passphrase. Also, it is recommended that you set the `DM_CRYPTO_FILE` environment variable to the location of the AEK and configure each product to reference the location defined in the environment variable. (Refer to the individual product documentation for instructions.)

The AEK and distributed sites

If your repository is a distributed repository, all servers at all sites must be using the same AEK. After you install the remote sites, you must copy the AEK from the primary site to the same location in each the remote site.

In multiple-repository distributed sites, each repository maintains its own AEK. When users work across the repositories, the servers in each repository take care of any encryption or decryption requirements locally.

Backing up the AEK

It is strongly recommended that you back up the AEK file separately from the repository and content files. Backing up the AEK and the data it encrypts in different locations helps prevent a “dictionary-based” attack on the AEK.

Repository encryption keys

A repository encryption key is created for a repository when the repository is created. The key is encrypted using the AEK and stored in the `i_crypto_key` attribute of the `docbase` config object. This attribute cannot be changed after the repository is created.

Content Server uses the repository encryption key to create the signature on audit trail entries. The server also used the repository encryption key to encrypt file store keys.

Encryption utilities

There are three utilities that are used to manage the AEK:

- `dm_crypto_boot`

Use this utility if you are protecting the AEK with a custom passphrase. Use it to:

- Install an obfuscated AEK or passphrase in the host machine's shared memory after you define a custom passphrase.
- Reinstall an obfuscated AEK or passphrase in the host machine's shared memory if you stop and restart a server host machine.
- (Windows only) Reinstall an obfuscated AEK or passphrase in the host machine's shared memory if you log off the host machine after you stop Content Server.
- Clean up the shared memory used to store the obfuscated AEK and passphrase.

It is not necessary to use this utility if you are protecting the AEK with the default, Documentum passphrase. Refer to [Using `dm_crypto_boot`, page 446](#), for more details and instructions on using the utility.

- `dm_crypto_create`

This utility is run automatically, during the Content Server installation process, to create and install the AEK. You can use this utility to determine if an AEK exists at a specified location and can be decrypted with a specified passphrase. Refer to [Troubleshooting with `dm_crypto_create`, page 447](#), for instructions.

- `dm_crypto_change_passphrase`

The `dm_crypto_change_passphrase` utility changes the passphrase used to encrypt an AEK. Refer to [Changing a passphrase, page 447](#), for instructions.

Using dm_crypto_boot

The dm_crypto_boot utility obfuscates the AEK or the passphrase used to decrypt the AEK and puts the obfuscated AEK or passphrase in shared memory. The utility is only used when you are protecting the AEK with a custom passphrase. It is not necessary if you are using the Documentum default passphrase. If you are protecting the AEK with a custom passphrase, you must run the utility in the following situations:

- When you stop and restart the host machine
After you stop a host machine, run this utility after restarting the host and before restarting the product or products that use the AEK.
- After you install the Content Server software and before you install the remaining products

You can also use this utility to clean up the shared memory region used to store the AEK.

To run this utility, you must be a member of the dmadmin group and you must have access to the AEK file.

The syntax for the utility is:

```
dm_crypto_boot[-passphrase passphrase] -location location|-all  
[-remove] [-help]
```

The -passphrase argument identifies the passphrase used to encrypt the AEK. If you do not include the argument or if you include the argument keyword but not its value, the utility prompts for the passphrase. (If the user is prompted for a passphrase, the passphrase is not displayed on screen when it is typed in.)

You must include either the -location or -all argument. Use -location if only one product is accessing the AEK. Using -location installs an obfuscated AEK in shared memory. The -location argument identifies the location of the AEK in the installation. If you don't include the argument, the utility looks for the location defined in DM_CRYPTO_FILE. If that environment variable is not defined, the utility assumes the location %DOCUMENTUM%\dba\secure\aekey.

Use the -all argument if there are multiple products on the host machine that use the same passphrase for their AEKs. If you include -all, the utility obfuscates the passphrase and writes it into shared memory instead of the AEK. Each product can then obtain the passphrase and use it to decrypt the AEK for their product.

To clean up the shared memory used to store the obfuscated AEK or passphrase, execute the utility with the -remove argument. You must include the -passphrase argument if you use -remove.

The -help argument returns help information for the utility. If you include -help, you cannot include any other arguments.

Troubleshooting with `dm_crypto_create`

You can run the `dm_crypto_create` utility with the `-check` argument to determine whether an AEK exists at a specified location and whether a particular passphrase can be used to decrypt it. The syntax is:

```
dm_crypto_create [-location location] [-passphrase passphrase|-noprompt] -check [-help]
```

The `-location` argument identifies the location of the AEK whose existence you wish to determine. If you don't include the argument, the utility looks for location defined in `DM_CRYPTO_FILE`. If that environment variable is not defined, the utility assumes the location `%DOCUMENTUM%\dba\secure\aek.key` (`$DOCUMENTUM/dba/secure/aek.key`).

The `-passphrase` argument identifies the passphrase whose use you wish to check. If you do not include the `-passphrase` argument, the utility assumes the default passphrase but prompts you for confirmation. Consequently, if you are checking the default passphrase and wish to avoid the confirmation request, include the `-noprompt` argument. (The `-passphrase` and `-noprompt` arguments are mutually exclusive.)

If the AEK file does not exist at the specified location, the utility returns 0. If the AEK exists but decryption fails, the utility returns 1. If the AEK exists and decryption succeeds, the utility returns 2.

The `-help` argument returns help information for the utility. If you include `-help`, you cannot include any other arguments.

Changing a passphrase

Use `dm_crypto_change_passphrase` to change the passphrase used to encrypt the AEK in the installation. Installing the Content Server software installs the AEK encrypted using a default passphrase. It is strongly recommended that you change the default to a custom passphrase of your choosing. Use this utility, also, if business rules or a security breach require you to change the passphrase. All Documentum products that use the affected AEK must be stopped before running the utility.

The syntax is:

```
dm_crypto_change_passphrase [-location location] [-passphrase [old_passphrase]] [-newpassphrase [new_passphrase]] [-noprompt] [-help]
```

The `-location` argument identifies the location of the AEK whose passphrase you wish to change. If you don't include the argument, the utility looks for location defined in `DM_CRYPTO_FILE`. If that environment variable is not defined, the utility assumes the location `%DOCUMENTUM%\dba\secure\aek.key`.

The `-passphrase` argument identifies the current passphrase associated with the AEK. The `-newpassphrase` argument defines the new passphrase you wish to use to encrypt the AEK. Both phrases interact in a similar manner with the `-noprompt` argument. The behavior is described in [Table 11-13, page 448](#).

Table 11-13. Interaction of `dm_crypto_change_passphrase` arguments

	-noprompt included	-noprompt not included
-passphrase argument not included	Utility prompts for a passphrase	Utility assumes the Documentum default passphrase
-passphrase keyword is included but no value	Utility prompts for a passphrase	Utility assumes the Documentum default passphrase
-newpassphrase argument not included	Utility prompts for a new passphrase	Utility assumes the Documentum default passphrase
-newpassphrase keyword is included but no value	Utility prompts for a new passphrase	Utility assumes the Documentum default passphrase

You must include a value for at least one of the passphrases. You cannot be prompted for both values or allow both values to default.

To illustrate the use of this utility, here are some examples. The first example changes the passphrase for the Content Server host AEK from the Documentum default to “`custom_pass1`”. The `-passphrase` argument is not included and the `-noprompt` is included, so the utility assumes that the current passphrase is the default passphrase.

```
dm_crypto_create -location %DOCUMENTUM%\dba\secure\aekey
-newpassphrase custom_pass1 -noprompt
```

This next example changes the passphrase from one custom passphrase to another:

```
dm_crypto_create -location %DOCUMENTUM%\dba\secure\aekey
-passphrase genuine -newpassphrase glorious
```

This final example changes the passphrase from a custom passphrase to the default:

```
dm_crypto_create -location %DOCUMENTUM%\dba\secure\aekey
-passphrase custom_pass1 -noprompt
```

Because the new passphrase is set to the Documentum default passphrase, it isn’t necessary to include the `-newpassphrase` argument. The utility assumes that the new passphrase is the default if the argument is not present and the `-noprompt` argument is present.

The `-help` argument returns help information for the utility. If you include `-help`, you cannot include any other arguments.

Managing the login ticket key

The login ticket key is used to generate and validate login tickets and application access control tokens. The key is installed automatically when a repository is created. Each repository has one login ticket key. (For a description of login tickets and how login ticket keys are used, refer to [Login tickets, page 35](#), in *Content Server Fundamentals*. [Application access control tokens, page 39](#), also in *Content Server Fundamentals*, describes application access control tokens.)

Exporting and importing a login ticket key

If you are using global login tickets or global application access control tokens, both the repository generating the ticket or token and the repository accepting the ticket or token must be using the same login ticket key. When you install a repository, the installation configures a login ticket key for the repository. However, each key is unique. Consequently, to use global login tickets or tokens, you must export a login ticket key from one repository among those that will be exchanging global login tickets or tokens and import that key into the other repositories exchanging global tickets or tokens.

To export a login ticket key, use the EXPORT_TICKET_KEY administration method. Use Documentum Administrator to execute these methods. These methods are also available as DFC methods (exportTicketKey and importTicketKey) in the IDfSession interface.

Resetting a login ticket key

Resetting a login ticket key replaces a repository's current login ticket key with a newly generated key. You may need to reset a login ticket key if the current key is compromised. If you reset the login ticket key, the repository's server will not accept any login tickets generated using the old key.

To reset a login ticket key, execute the RESET_TICKET_KEY method. Use Documentum Administrator to execute that method. You can also use the DFC method, resetTicketKey, in the IDfSession interface.

Configuring a repository's trusted repositories

A repository that accepts a global login ticket or global application access control token must trust the repository in which the login ticket or token was generated. To define

which repositories are trusted, you must define the repository's trust mode. You can configure a repository to:

- Trust all other repositories
- Trust only specifically identified repositories

Use Documentum Administrator to define the trust mode for a repository. If you choose to allow the repository to trust all other repositories, the choice is recorded in the `trust_by_default` attribute (in the `docbase` config object), which is set to `TRUE`.

If you choose to allow the repository to trust only a specific set of repositories, the `trust_by_default` attribute is set to `FALSE`, and you must provide the names of the trusted repositories. Those names are recorded in the `trusted_docbases` attribute of the `docbase` config object.

For detailed instructions, refer to the Documentum Administrator online help. For an overview of trusted repositories and how they function with login tickets and tokens, refer to [Trusted repositories](#), page 37 in *Content Server Fundamentals*.

Configuring login ticket use

Use the following procedures to customize login tickets at your site.

Configuring the default login ticket timeout

Login tickets are only valid for a specified period of time. A ticket's validity period is defined by either:

- An argument when the ticket is generated
- The value in the `login_ticket_timeout` attribute of the server issuing the method to generate the ticket

If you do not specify a validity period when a ticket is generated, the period defaults to the value defined in the `login_ticket_timeout` attribute. The value of that attribute is set to 5 minutes by default when a repository is installed.

Neither the attribute value nor the argument value can exceed the maximum interval defined in the `max_login_ticket_timeout` attribute defined in the server's server config object. This attribute is set to 43200 minutes (30 days) by default when a repository is installed. (If the argument exceeds the maximum, the argument value is ignored and the maximum value is used.)

To reset either or both the `login_ticket_timeout` and `max_login_ticket_timeout` attributes, use Documentum Administrator. Refer to the Documentum Administrator's online help if needed.

Restricting a Superuser's use of global tickets

You may want to restrict superusers from using global login tickets to ensure that a user in one repository cannot use a global login ticket to gain superuser privileges in another repository. ([Restricting superuser use, page 39](#), in *Content Server Fundamentals* describes this possible situation.)

The restriction is configured at the server level. You can decide on a server-by-server basis, for each repository, whether the server can accept global login tickets for superusers in the repository. Use Documentum Administrator to record the choice for each server. The choice is recorded in the `restrict_su_ticket_login` attribute of the server config object.

Revoking tickets for a specific repository

It is possible to set a cutoff date for login tickets accepted by a repository. If you do so, the repository's servers consider any login ticket issued before that date as invalid, and the servers refuse to accept them.

Use Documentum Administrator to set a login ticket cutoff date for a repository. The date is recorded in the `login_ticket_cutoff` attribute of the doabase config object.

Configuring application access control token use

Application access control (AAC) tokens are a security feature that you can use to control access to a repository based on who is requesting access, the application or host from which the request is issued, or some combination of these factors.

AAC tokens are enabled at the server level, to give you flexibility in designing your security. For example, you can set up a server that requires a token to service users outside a firewall and another server that does not require a token for users inside the firewall.

Tokens are used in addition to user names and passwords (or login tickets) when issuing a connection request. They are not a substitute for either. If a server requires a token and the user making the connection request is not a superuser, the connection request must be accompanied by a token. Only superusers can connect to a repository through a server requiring a token without a token. For complete information about tokens and how they are implemented and used, refer to [Application access control tokens, page 39](#), in *Content Server Fundamentals*.

Enabling AAC token use by a server

Use Documentum Administrator to enable the use of application access control tokens by a particular server. If use is enabled, non-superuser users cannot access the repository through that server unless the constraints specified in the token are satisfied. Whether a server has AAC token use enabled is recorded in the `application_access_control` attribute of its server config object.

Enabling token retrieval by the client library

You can configure client library behavior to allow the client library to automatically retrieve a token from storage and append that token to an application's connection request if a token is required and none is provided in the connection request. If retrieval is enabled, the client library searches for a token file whose name matches the name of the repository specified in the connection request. If such a token file is found, the token it contains is appended to the connection request. If such a token file is not found, the client library appends the token from the token file named `default.tkn`, if one is available.

This feature helps ensure that the appropriate token is provided when an application is run from different machines and that older applications can connect to a server that requires an AAC token for connection.

To enable token retrieval:

1. Generate the token files using `dmtkgen`.
2. Configure the retrieval behavior in the `dmcl.ini` file.
 - a. Set `token_storage_enabled` to T.
 - b. Ensure that `token_storage_path` is set to the desired token storage directory.

Tokens in storage are found in token files generated by the `dmtkgen` utility. For instructions on using the utility, refer to [Generating tokens for storage, page 453](#).

Retrieving the tokens is configured using two `dmcl.ini` keys: `token_storage_enabled` and `token_storage_path`.

The `token_storage_enabled` key is a Boolean key that controls whether the client library can retrieve tokens from storage. If it is set to T, the client library will attempt to retrieve a token from storage for use when a connect request without a token is issued to a server requiring a token. If the key is set to F, the client library does not retrieve tokens from storage.

The `token_storage_path` key tells the client library where to find the token files generated by the `dmtkgen` utility. Any tokens that you generate using `dmtkgen` must be placed in this location. If they are not, the client library will not find them.

When you install the DFC on a client host machine for the first time, the installation sets the `token_storage_enabled` key to F and the `token_storage_path` to `<user-selected drive>:\Documentum\apptoken`. If you upgrade or reinstall the DFC, the procedure will not reset those keys if you have changed them.

When you install Content Server installation for the first time on a host machine, the process also installs the DFC. The DFC installation process sets the `token_storage_enabled` key in the `dmcl.ini` on the server host to F and the `token_storage_path` to `%Documentum%\apptoken` (`$DOCUMENTUM/apptoken`). However, when the Content Server installer runs, it resets the `token_storage_enabled` key to T. The `token_storage_path` is not reset.

The `dmcl.ini` file on the server host is used by the internal methods to connect to repositories. The `token_storage_enabled` and `token_storage_path` settings in this `dmcl.ini` file affect methods associated with `dm_method` objects. Replication and federation methods are not affected by the settings in this `dmcl.ini` file because these jobs execute as a superuser. (For more information about how internal methods are affected by this setting, refer to [Internal methods, user methods, and tokens, page 43](#), in *Content Server Fundamentals*.)

If you are installing an upgrade to Content Server or a previous DFC installation occurred and a `dmcl.ini` file already exists on the host machine with these keys set, installing Content Server and a new DFC does not overwrite their values.

Generating tokens for storage

Use the `dmtkgen` utility to generate application access control tokens to be stored on host machines. This utility is found in `%DM_HOME%\bin` (`$DM_HOME/bin`). Each execution of the utility creates one token file in XML format. The file is an ASCII file. The file contains the token and the information used to generate the token. Here is a sample of the file's format:

```
<token>
<docbase>mytestdb</docbase>
<user>me</user>
<scope>global</scope>
<timeout>40000</timeout>
<appidhash>hash_of_application_id</appidhash>
<machineonly>T</machineonly>
<tokendata>DM_TOKEN=tokenstring</tokendata>
</token>
```

[Table 11-14, page 454](#), lists the arguments accepted by this utility.

Table 11-14. dmtkgen utility arguments

Argument	Description
-u <i>username</i>	User as whom the utility is running This is a required argument.
-p <i>password</i>	Password for the user identified in <i>username</i> This is a required argument.
-d <i>domain</i>	Domain of the user identified in <i>username</i> This argument is required only if a domain is required to authenticate the user.
-b <i>repository_name</i>	Name of the repository to which to connect to create the token. This is a required argument.
-a <i>user group name</i>	Name of the user or group who can use this token. If a group is specified, all members of the group can use this token. This is an optional argument. If unspecified, then all users can use the token.
-s <i>scope</i>	Scope of the generated token. Valid values are <ul style="list-style-type: none"> • <code>docbase</code> Specifying <code>docbase</code> restricts the token's use to the repository identified in the output file's name. • <code>global</code> Specifying <code>global</code> allows the token to be used to connect to any repository having the same LTK as the repository in which the token was generated. This is an optional argument. The default is <code>global</code> .
-t <i>timeout</i>	Validity period for the generated token. The value is interpreted in minutes. This is an optional argument. The default is one year, expressed in minutes.
-i <i>application_identifier</i>	User-defined application identifier representing the application for which this token is valid. This is an optional argument. If unspecified, the token is valid for all applications.

Argument	Description
<code>-m <i>machine_only</i></code>	<p>Boolean flag indicating whether the token is valid only when used on the machine on which the token was generated. T means the token may only be used from the machine on which it was generated. F means that the token may be sent from any machine.</p> <p>This is an optional argument. The default is F.</p>
<code>-o <i>output_file</i></code>	<p>The name of the generated XML token file. You can specify a full file path or only the file name.</p> <p>The file name must be either:</p> <pre>default.tkn</pre> <p>or</p> <pre>repository_name.tkn</pre> <p>This is an optional argument. It defaults to <code>repository_name.tkn</code> where <code>repository_name</code> is the repository identified in the <code>-b</code> argument. If you include a name but not a file path, the file is stored in the current directory.</p> <p>The implementation imposes a naming constraint on global tokens. Refer to Naming the output file, page 455, for information.</p>

Naming the output file

Token files are retrieved by repository name. If the client library is looking for a token in storage for a particular connection request, it searches for a token whose name is the name of the requested repository in the Connect method. If the library cannot find a token whose name matches the repository name in the connection request, it searches for a token file called `default.tkn`. Because of this token file search algorithm, if you create a global token file for use across multiple repositories, you must name the file `default.tkn`.

Storing tokens generated by `dmtkgen`

The client library looks for the stored tokens in the location identified by the `token_storage_path` key in the client's `dmcl.ini` file. After you generate the token file, you can copy the file to that location for each client machine where the token will be

used. Because the generated file is an ASCII file, you can copy the file across operating systems. For example, if it was generated on a Windows host machine, you can copy it to a UNIX host machine. Similarly, if it was generated on a UNIX host machine, you can copy it to a Windows host machine.

If the location specified in `token_storage_path` is visible to the machine on which you generate the token, you can specify the location in the `-o` argument and the token file will be saved to the correct location automatically.

The `token_storage_path` attribute is set automatically when the DFC is installed. For more information about its setting, refer to [Enabling token retrieval by the client library](#), page 452.

Tools and Tracing

This chapter contains information about the automated tools you can use to manage and monitor repositories and associated file systems. The chapter includes the following topics:

- Essential tool concepts, page 458
- Activating and scheduling administration tools, page 465
- Running administration jobs on demand, page 465
- Archive , page 466
- Audit Management , page 468
- Consistency Checker, page 471
- Content Replication, page 477
- Content Warning , page 480
- Create Full-Text Events, page 482
- Data Dictionary Publisher, page 487
- Database Space Warning , page 489
- Dm_LDAPsynchronization, page 491
- Dmclean , page 494
- Dmfilescan , page 499
- File Report , page 504
- Group Rename, page 509
- Index Agent Startup, page 510
- Log Purge , page 510
- Queue Management , page 514
- Remove Expired Retention Objects, page 517
- Rendition Manager , page 520
- State of the Repository Report , page 525
- Swap Info , page 529
- ToolSetup, page 530

- [Update Statistics](#) , page 530
- [User Chg Home Db](#), page 533
- [User Rename](#), page 534
- [Version Management](#) , page 536

The chapter also provides maintenance and troubleshooting information for the tools. Refer to [Tool maintenance and troubleshooting](#), page 540.

Tracing facilities provide diagnostic and troubleshooting information. Refer to [Tracing](#), page 541, for information about the server tracing facilities.

Essential tool concepts

Documentum provides a suite of automated tools installed with Content Server that can simplify your system administration tasks. For example, the Database Space Warning tool helps prevent unexpected downtime by warning you when space on a storage disk is running low. The Full Text Index Management tool automatically updates full text indexes, freeing you from that task. Each tool identifies a job to run on a specific schedule.

The tools are described briefly in [Table 12-1](#), page 458. Typically, the tools are managed by the Documentum system administrator. However, some tools might be managed more efficiently by another person. For example, you may want the RDBMS administrator to manage the Database Space Warning tool because this tool monitors space and fragmentation in the underlying database.

Documentum Administrator provides a graphical interface for defining, modifying, and monitoring the tools and their associated jobs.

Table 12-1. EMC Documentum tool suite

Tool	Description
Archive	Automates archive and restore between content areas
Audit Management	Deletes unneeded audit trail objects
Consistency Checker	Checks the repository for consistency across a variety of object types.
Content Replication	Automates replication of document content when using distributed filestores

Tool	Description
Content Storage Warning	Monitors disk space on the devices used to store content and index files. Queues a warning message when a disk used for content and index storage reaches a user-defined percentage of capacity
Create Full-Text Events	Creates events for objects not indexed because the objects were created or modified between the creation of a full-text index and when the repository was upgraded. Can optionally be used to generate events to fully reindex a repository.
Database Space Warning	Monitors the RDBMS. Queues a warning message when the RDBMS reaches a user-defined percentage of capacity. Note: This tool is not installed for installations running against MS SQL Server or DB2.
Dm_LDAPSynchronization	Determines all changes in the LDAP directory server information and propagates the changes to the repository.
Dmclean	Automates the dmclean utility, which removes orphaned content objects, notes, ACLs, and SendToDistribution workflow templates from a repository
Dmfilescan	Automates the dmfilescan utility, which removes orphaned content files from a specified storage area of a repository
DistOperations	Performs the distributed operations necessary to manage reference links. This is an internal job that is installed as part of the tool suite but is not available for users to manipulate or change. Consequently, it is not described in detail in this chapter. Refer to the <i>Distributed Configuration Guide</i> for more details.
File Report Utility	Provides report to help restore deleted or archived document content using your file system backups. This is a method that applies only when using distributed storage areas.

Tool	Description
Log Purge	Deletes log files for the server, session, connection broker, and agent and the trace log files resulting from method and job executions
Queue Management	Deletes dequeued objects in the dmi_queue_item tables
Rendition Management	Manages the removal of unwanted renditions of a document
State of the Repository Report	Provides information about the status of the Documentum environment
Swap Info	Reports on swap space availability and usage Note: The Swap Info tool is not installed if the Content Server is installed on an HPUX platform.
ToolSetup	Installs system administration tools for both primary (RDBMS) site and remote sites
Update Stats	Updates stored statistics on the underlying RDBMS tables, to aid in query performance.
Version Management	Manages the removal of unwanted versions of documents

How tools are implemented

Most tools are implemented as jobs, which are objects that reference a method object and have attributes that define an execution schedule for the method. The methods associated with the system administration tools call Docbasic procedures. (Refer to [Chapter 4, Methods and Jobs](#), for more information about jobs and methods.)

Jobs are automatically executed on the schedule defined by their attributes. A server process named agent exec regularly checks the jobs in a repository and runs those that are ready for execution. A job is ready for execution when its scheduled next run time is less than or equal to the current time. Jobs can also be run on demand. (Refer to [Activating and scheduling administration tools](#), page 465, for instructions on setting job schedules. Information about running tools on demand is in [Running administration jobs on demand](#), page 465.)

ToolSetup is an exception in that it is not implemented as a job. ToolSetup is the utility that installs the tool suite. The utility is integrated into dm_configure for installations in

either a single node environment or in a distributed environment. However, you can also run the utility manually if needed.

Standard arguments

When the agent exec launches a job, it passes the four standard arguments listed in [Table 12-2, page 461](#), to the job's method.

Table 12-2. Standard arguments passed to jobs

Argument	Datatype	Value	Description
DOCBASE_NAME	string(64)	<i>repository_name</i>	Identifies the repository
USER_NAME	string(64)	<i>user_name</i>	The user of the tool
JOB_ID	ID	<i>job_object_id</i>	Object ID of the job
METHOD_TRACE_LEVEL	integer	<i>level</i>	Method trace level to use. The default is 0 (no tracing).

Each job-implemented tool also has arguments which are defined in the `method_arguments` attribute for the job. These argument values are obtained from the job object (using the job ID value) when the job executes. Refer to the individual tool descriptions for information about the tool-specific arguments.

The QUEUEPERSON argument

Many tools accept an argument called `-queueperson`. This argument defines which repository user receives the Inbox and email notifications sent by the tools. If you do not define `-queueperson` (in the `method_arguments` attribute), the Inbox and email notifications are sent to the user identified in the `operator_name` attribute of the server's server config object. (That attribute is set to the name of the repository owner by default.) However, you may want to change this for some tools. For example, the Database Space Warning tool provides status information about the RDBMS, and you may want its notifications sent to the RDBMS DBA.

The window interval

When you restart a server, the tools scheduled to run while the server was shut down will not necessarily run immediately. Some tools have a default window on either side of the scheduled run time that allows the tool schedule to recover gracefully. For example, suppose that a tool has a `window_interval` of 120 minutes (2 hours) and it is normally supposed to execute at 9 p.m. The tool will only be able to run between the hours of 7 p.m. and 11 p.m. If it is started before 7 p.m., it aborts and is rescheduled to run at 9 p.m. the same day. If it is started after 11 p.m., it aborts and is rescheduled to run at 9 p.m. the next day.

To illustrate, suppose a tool is scheduled to execute daily at 11 p.m. and that the server is shut down from 9 a.m. on July 21 to 9 a.m. on July 22. When you restart the server at 9 a.m. on the July 22, the server attempts to execute the tool. However, because 9 a.m. is not within the tool's window, the tool aborts the July 21, 11 p.m. job, and is rescheduled to run at 11 p.m. that night (July 22).

This window of execution is provided for most tools to ensure that server start-ups are not delayed by the execution of tools. You can change the size of the window by setting the `-window_interval` argument for the job (in the `method_arguments` attribute). This argument takes an integer value that expresses the length of the desired interval in minutes. For example if you wanted to reset the interval to 1 hour on either side of the job's scheduled execution time, you would set `-window_interval` to 60 for that job.

Refer to the description of each tool to determine if the tool takes the `-window_interval` argument and, if so, the argument's default value.

Reports and trace log files

All jobs in the tool suite generate reports. Trace log files are only generated if tracing is turned on for the job.

Reports

The job reports summarize the results of the job in an easily interpreted format. The reports are named for the tool that creates them. In a distributed environment, the report for the primary site is named with the tool's name and the reports for the remote sites have the site's server config name appended to the tool's name. The following tools generate site-specific reports in a distributed environment:

- ContentWarning
- Dmclean

- Dmfilescan
- LogPurge
- SwapInfo

On Windows, job reports are returned using the client code page. (The client code page is identified in the client's dmcl.ini file. If unspecified in the dmcl.ini file, the server chooses a default based on the client's locale. The defaults are those listed for default_client_codepage in [Table 3-1, page 87.](#))

On UNIX, job reports are returned using the UTF-8 code page.

You can view job reports through Documentum Administrator. To see examples of some of the reports, refer to the descriptions of the individual tools.

The content files for job reports are stored in the default storage area for SysObjects. However, job reports are not indexed.

Storage

In the repository, job reports are stored in /System/Sysadmin/Reports. Each time a job executes, the job report in this location is versioned.

The reports are also stored in the file system in %DOCUMENTUM%\dba\log\repository_id\sysadmin (\$DOCUMENTUM/dba/log/repository_id/sysadmin). The reports in this directory are overwritten each time the job executes.

The job reports are also stored in %DOCUMENTUM%\share\temp\ldif\repository_name (\$DOCUMENTUM/share/temp/ldif/repository_name) for access through Documentum Administrator.

Trace log files

Trace logs contain status information logged by the job and the text of the report. The trace level in effect determines how much information is logged. For example, the trace level 4 directs the system to log a medium amount of status information and a trace level of 10 provides verbose debug-level tracing. A trace level of 0 turns off tracing. The default trace level for all jobs in the tool suite is 0.

To turn on tracing and generate a trace log file, set the method_trace_level argument to one of the valid tracing levels.

To view a trace log file, use Documentum Administrator. Viewing a trace log file is available in Job Management.

Note: If the dmcl.ini file used by the Content Server executing a job has the trace_file key set, the tracing information generated by the job is recorded in the location defined

by that key. When that occurs, Documentum Administrator cannot display the trace file for the job. Content Server uses the dmcl.ini file found in %DOCUMENTUM% (\$DOCUMENTUM) on the server's host machine.

Storage

In the repository, job log files are stored in /Temp/Jobs/*tool_name*Trace. The file is named for the tool that generated it. For example, the job log file for the Rendition Management tool is called RenditionMgtTrace. The log files in this directory are versioned when the jobs execute.

The log files are also stored in the file system in %DOCUMENTUM%\dba\log\repository_*id*\sysadmin (\$DOCUMENTUM/dba/log/repository_*id*/sysadmin). The log file in this directory is overwritten each time the job executes.

The job log files are also stored in %DOCUMENTUM%\share\temp\ldif\repository_*name* (\$DOCUMENTUM/share/temp/ldif/repository_*name*) for access through Documentum Administrator.

Email messages

Tools can generate an email message in addition to a report. Messages are generated if an error occurs while the tool is executing or if a warning tool (such as Content Warning) encounters a condition that requires a warning.

The message is sent to the queueperson for the tool described above. It identifies the repository, the event ID, the trace log file name, who sent the message, and the task name. This message may also contain a warning.

For example, here is an email message sent by the Content Warning tool to the repository owner:

```
X-UIDL: 827528501.002
Date: Fri, 22 Mar 1996 12:02:34 +0800
From: test1@lapdog (Trashable Repository - SunOS5)
To: stevek@lapdog
Subject: Event storage_01 has occurred on ContentWarning.Result
(090007d080045f8d) by testsol
REPOSITORY:      test1
EVENT:           storage_01
NAME:            ContentWarning.Result
SENT BY:         test1
TASK NAME:       event
Take a look at /export/nfs1-4--it's 90% full!!!
```

Activating and scheduling administration tools

Before a tool can be executed, it must be in an active state and it must have an execution schedule.

Activation

Some of the tools are installed in the active state and some are installed in the inactive state. For example, the tools that remove objects from the respository or file system, such as the Version Management and Rendition Management tools, are installed in the inactive state because they require you to set arguments that define what you want to remove. Consequently, they are installed in the inactive state so you can set the arguments before activating them.

Use Documentum Administrator to activate (or inactivate) an administration job. After a tool is activated, it is executed at the interval defined in its schedule.

Defining job schedules

Each administration job is installed with a default schedule. Typically, the default schedules run a job once a day or once a week. You can change the schedules to fit the needs of your site.

You can schedule a job to run independently of other jobs or include it in a job sequence. A job sequence identifies a set of jobs that are dependent on one or more jobs within the sequence. The dependent jobs cannot be executed until the job or jobs on which they depend are completed successfully. Both the dependent jobs and the jobs on which they depend are included in a job sequence.

To view or reset a job's schedule or to create a job sequence, use Documentum Administrator. For information about setting schedules, refer to the Documentum Administrator online Help or to [Scheduling jobs, page 148](#). Refer to [Introducing job sequences, page 143](#) for more information about job sequences.

Running administration jobs on demand

You can execute a tool on demand. For example, after a project is completed, you may want to run the Version Management tool to remove old versions of the project's documentation, using the tool's `custom_predicate` attribute to constrain the search to old

versions. Or, you may want to run the Swap Space Info tool during a peak usage time, to see how swap space is being affected.

You can run a tool on demand at any time within its window interval. You cannot run a tool on demand outside of its window interval. (Refer to [The window interval, page 462](#), for information about the window interval.)

Running a tool on demand does not affect its normal schedule. A tool does not need to be active for you to run it on demand. Use Documentum Administrator to run a tool on demand.

Archive

The Archive tool automates archive and restore between content areas. Archive older or infrequently accessed documents to free up disk space for newer or more frequently used documents. Restore archived documents to make the archived documents available when users request them. For complete information on configuring archiving, refer to [Archiving and restoring documents, page 280](#).

The Archive tool is active by default, and runs once daily.

Arguments

The Archive tool has nine arguments, described in [Table 12-3, page 466](#).

Table 12-3. Archive arguments

Argument	Datatype	Default	Description
<code>-docbase_name</code> <i>repository</i>	string(64)	-	Identifies the repository that contains the document or documents to archive. Use the repository's name.
<code>-Username</code>	string(64)	-	Identifies the user executing dmarchive. If unspecified, the current user is assumed.
<code>-Ppassword</code>	string(64)	-	Password for the user executing dmarchive. If unspecified, the user is prompted.

Argument	Datatype	Default	Description
-archive_dir <i>directory</i>	string	-	The location of the archive directory. Use a full path specification.
-queue_name <i>name</i>	string	-	Identifies the repository operator. Specify the operator's repository user name. If unspecified, the tool assumes the user named in the operator_name attribute of the server config object.
-queue_event <i>event_id</i>	ID	-	Object ID of the queue item object representing an archive or restore event. If set, the tool processes only the specified event.
-do_archive_events	Boolean	-	TRUE directs the tool to process only archive events.
-do_restore_events	Boolean	-	TRUE directs the tool to process only restore events.
-verbose	Boolean	T	TRUE directs the tool to print trace messages.
-window_interval	integer	720	Defines window in which the tool can run. Value is interpreted in minutes.
-queueperson	string	-	Identifies the user who receives Inbox and email notifications from the tool.

Guidelines

The Archive tool puts all the documents it is archiving in one dump file. This means you must move the entire dump file back to the archive directory to restore a single document in the file. If the files are extremely large, this can be a significant performance hit for restore operations.

Audit Management

The Audit Management tool deletes audit trail entries. When an audited event occurs, an audit trail entry is created for that event. If the audit trail entries are not removed periodically, the tables for the `dm_audittrail` object type can grow quite large, and performance degrades when audited events occur. The Audit Management tool automates the task of removing unneeded audit trail objects.

The tool executes the `dm_AuditMgt` job. The job uses the `PURGE_AUDIT` administration method to remove the audit trail entries from the repository. Consequently, to use this tool, the Documentum installation owner must have Purge Audit privileges. (The tool runs under the Documentum installation owner's account.) All executions of the tool are audited. The generated audit trail entry has the event name `dm_purgeaudit`.

Which audit trail objects to remove is determined by the `cutoff_days` and `custom_predicate` arguments. The `cutoff_days` argument specifies the age of the objects to delete. The `custom_predicate` argument is then applied to those items meeting the age requirement.

By default, the `cutoff_days` argument is set to 90 and the `custom_predicate` argument is set to remove only audit trail objects generated by system-defined events. (The tool does not delete audit trail objects generated by user-defined events by default.)

To change the age cutoff, reset the `cutoff_days` argument.

To choose the objects to remove from the subset selected by `cutoff_days`, change the `custom_predicate` argument. By default, the custom predicate includes three conditions:

- `delete_flag=TRUE`
- `dequeued_date=value` (*value* is computed using the `cutoff_days` argument)
- `r_gen_source=1`

You cannot change the first two conditions. The third condition, `r_gen_source=1`, directs the server to delete only audit trail objects generated by system-defined events. If you want to remove only audit trail objects generated by user-defined events, reset this to `r_gen_source=0`. If you want to remove audit trail objects generated by both system- and user-defined events, remove the `r_gen_source` expression from the custom predicate.

You may also add other conditions to the default custom predicate. If you add a condition that specifies a string constant as a value, you must enclose the value in two single quotes on each side. For example, suppose you want to remove only audit trail entries that record `dm_checkin` events. To do so, add the following to the `custom_predicate`:

```
event_name='dm_checkin'
```

`dm_checkin` is enclosed by two single quotes on each side. Do not use double quotes. These must be two single quotes.

The Audit Management tool generates a status report that lists the deleted `dm_audittrail` entries. The report is saved in the repository in `/System/Sysadmin/Reports`.

If an error occurs while the tool is executing, the server sends email and inbox notification to the user specified by the `-auditperson` argument.

The Audit Management tool is installed in the inactive state. The first time you execute the tool, it may take a long time to complete.

Arguments

The Audit Management Tool has four arguments, described in [Table 12-4, page 469](#).

Table 12-4. Audit Management arguments

Argument	Datatype	Default	Description
<code>-window_interval</code>	<i>integer</i>	120	Defines the execution window for the tool. Value is interpreted in minutes.
<code>-queueperson</code>	<code>string(32)</code>	-	User who receives email and inbox notifications from the tool. The default is the user specified in the <code>operator_name</code> attribute of the server config object.
<code>-custom_predicate</code> <i>qualification</i>	<code>string</code>	-	<p>A WHERE clause <i>qualification</i> for the query that selects audit trail entries for deletion.</p> <p>The qualification must be a valid qualification and can reference only audit trail object type attributes. For example, a valid qualification is <code>event='approved'</code> or <code>name='dadmin'</code>. (Refer to the general discussion of the tool, above, for details of setting this argument.)</p> <p>Refer to The WHERE clause, page 142 in the <i>Content Server DQL Reference Manual</i> for</p>

Argument	Datatype	Default	Description
-cutoff_days	<i>integer</i>	90	<p>complete information about WHERE clause qualifications.</p> <p>A minimum age, in days, for objects to delete. All audit trail objects older than the specified number of days and that meet the specified qualification are deleted.</p> <p>To delete all audit trail objects, set this value to zero (0).</p>

Guidelines

Audit trail entries are the result of audited events. The more events you audit, the more audit trail entries are generated in a fixed period of time.

You must decide if there are any reasons to keep or maintain audit trail entries. For example, you may want to keep certain items for traceability purposes. If so, leave this tool inactive or set the `cutoff_days` argument to a value that will save the audit trail items for a specified length of time.

After you have made your decisions, formulate a scheduling plan.

If you do not supply a value for `custom_predicate` or `cutoff_days`, all system-generated `dm_audittrail` entries older than 90 days are deleted.

Report sample

Here is a sample of the report generated by the Audit Management Tool.

```
AuditMgt Report For repository BLD9A As Of 10/19/98 12:26:31 PM
```

```
Parameters for removing audit trail items:
```

```
-----
- No items audited before 90 days will be removed...
- There is no custom predicate...
```

```
Looking for audit trail items to delete...
Destroying audit trail item with ID 5f00010080000124
Destroying audit trail item with ID 5f00010080000125
Destroying audit trail item with ID 5f00010080000126
```

```
Destroying audit trail item with ID 5f00010080000127
Destroying audit trail item with ID 5f00010080000128
Destroying audit trail item with ID 5f00010080000129
Destroying audit trail item with ID 5f0001008000012a
Destroying audit trail item with ID 5f0001008000012d
Destroying audit trail item with ID 5f0001008000012e
Destroying audit trail item with ID 5f0001008000012f
Destroying audit trail item with ID 5f00010080000130
Destroying audit trail item with ID 5f00010080000131
Destroying audit trail item with ID 5f00010080000132
Destroying audit trail item with ID 5f00010080000133
Destroying audit trail item with ID 5f00010080000134
Destroying audit trail item with ID 5f00010080000135
Destroying audit trail item with ID 5f00010080000136
Destroying audit trail item with ID 5f00010080000137
18 audit trail items deleted...
```

```
End of Audit Trail Management Report
Report End 10/19/98 12:26:33 PM
```

Consistency Checker

The Consistency Checker tool scans the repository and reports any inconsistencies such as type or object corruption, objects that reference a user, group, or other object that is non-existent in the repository and so forth. The tool does not attempt to fix any of the inconsistencies. Contact Documentum Technical Support for assistance in correcting errors in your repository found by the consistency checker.

[Appendix A, Consistency Checks](#), lists the consistency checks conducted by the tool and the error number assigned to each.

The job generates a report that lists the categories checked and any inconsistencies found. The report is saved to the repository in `/System/Sysadmin/Reports/ConsistencyChecker`. If no errors are found, the current report overwrites the previous report. If an error is found, the current report is saved as a new version of the previous report.

It is recommended that you run this tool on a repository before upgrading the repository to a new version of the Documentum Server.

The Consistency Checker job is active by default, running once a day.

Running the job from a command line

The Consistency Checker job is implemented as a script called `consistency_checker.ebs`. You can run the script manually, from the operating system prompt. The syntax is:

```
dmbasic -fconsistency_checker.ebs -eEntry_Point --repository_
name superuser password
```

repository_name is the name of the repository against which you are running the consistency checker, *superuser* is the user name of a repository superuser, and *password* is the password for the superuser's account.

When you run the consistency checker from the command line, the results of the checks are directed to standard output.

Arguments

The Consistency Checker job has no arguments.

Report sample

Here is a sample of a Consistency Checker report. This run of the tool found X inconsistencies.

```
Beginning Consistency Checks.....
```

```
Repository Name:  buzzard
Server Version:  5.1.0.63 Win32.SQLServer
Database:        SQLServer
```

```
#####
##
## CONSISTENCY_CHECK: Users & Groups
##
##      Start Time: 09-10-2002 10:15:55
##
##
#####
```

```
Checking for users with non-existent group
WARNING CC-0001: User 'docu' belongs to
non-existent group ''
WARNING CC-0001: User 'enr' belongs to
non-existent group ''
WARNING CC-0001: User 'marketing' belongs to
non-existent group ''
WARNING CC-0001: User 'nagboat' belongs to
non-existent group ''
WARNING CC-0001: User 'admingroup' belongs to
non-existent group ''
Rows Returned: 5
```

```
Checking for users belonging to groups not in dm_user
Checking for users not listed in dmi_object_type
Checking for groups not listed in dmi_object_type
Checking for groups belonging to non-existent groups
Checking for groups with non-existent super groups
#####
```

```
##
##
## CONSISTENCY_CHECK: ACLs ##
##
##      Start Time: 09-10-2002 10:15:55
##
##
#####

Checking for ACLs with non-existent users
Checking for ACLs with missing dm_acl_r table entries
Checking for sysobjects with acl_domain set to
  non-existent user
Checking for sysobjects that belong to
non-existent users
Checking for sysobjects with non-existent ACLs
Checking for ACL objects with missing dm_acl_s entry
Checking for ACL objects with r_accessor_permit
value but missing r_accessor_name value
Checking for ACL objects with r_accessor_name value
  but missing r_accessor_permit value
Checking for ACL objects with r_is_group value but
  missing r_accessor_permit value
Checking for ACL objects with r_is_group value but
  missing r_accessor_name value
Checking for ACL object with r_accessor_name value
  but missing r_is_group value
Checking for ACL object with r_accessor_permit value
  but missing r_is_group value

#####
##
##
## CONSISTENCY_CHECK: Sysobjects
##
##
##      Start Time: 09-10-2002 10:15:58
##
##
#####

Checking for sysobjects which are not referenced in
dmi_object_type
Checking for sysobjects that point to non-existent
  content
Checking for sysobjects that are linked to non-existent
  folders
Checking for sysobjects that are linked to non-existent
  primary cabinets
Checking for sysobjects with non-existent i_chronicle_id
Checking for sysobjects with non-existent i_antecedent_id
Checking for sysobjects with missing
dm_sysobject_r entries
Checking for sysobjects with missing
dm_sysobject_s entry

#####
##
##
```

```
## CONSISTENCY_CHECK: Folders and Cabinets
##
##      Start Time: 09-10-2002 10:16:02
##
##
#####
Checking for folders with missing dm_folder_r table
entries
Checking for folders that are referenced in dm_folder_r
but not in dm_folder_s
Checking for dm_folder objects that are missing an
entry in dmi_object_type
Checking for dm_folder objects that are missing
corresponding dm_sysobject entries
Checking for folders with non-existent ancestor_id
Checking for cabinet that have missing dm_folder_r
table entries
Checking for cabinets that are missing an entry in
dmi_object_type
Checking for folder objects with missing
dm_sysobject_r entries
Checking for folder objects with null r_folder_path
#####
##
##
## CONSISTENCY_CHECK: Documents
##
##
##      Start Time: 09-10-2002 10:16:03
##
##
##
#####
Checking for documents with a dm_sysobject_s entry
but no dm_document_s entry
Checking for documents with missing dm_sysobject_s
entries
Checking for documents with missing dmi_object_type
entry
#####
##
##
## CONSISTENCY_CHECK: Content
##
##      Start Time: 09-10-2002 10:16:03
##
##
##
#####
Checking for content objects that reference
non-existent parents
Checking for content with invalid storage_id
Checking for content objects with non-existent format
#####
##
```

```
##
## CONSISTENCY_CHECK: Workflow
##
##
##      Start Time: 09-10-2002 10:16:03
##
##
#####
Checking for dmi_queue_item objects with non-existent
  queued objects
Checking for dmi_workitem objects that reference
non-existent dm_workflow objects
Checking for dmi_package objects with missing
dmi_package_s entries
Checking for dmi_package objects that reference
  non-existent dm_workflow objects
Checking for workflow objects with non-existent
  r_component_id
Checking for workflow objects with missing
dm_workflow_s entry
Checking for work item objects with missing
dm_workitem_s entry

#####
##
##
## CONSISTENCY_CHECK: Types
##
##      Start Time: 09-10-2002 10:16:04
##
##
#####
Checking for dm_type objects with a non-existent
dmi_type_info object
Checking for dmi_type_info objects with a non-existent
  dm_type object
Checking for type objects with corrupted attribute
  positions
Checking for types with invalid attribute counts

#####
##
##
## CONSISTENCY_CHECK: Data Dictionary
##
##      Start Time: 09-10-2002 10:16:04
##
##
#####
Checking for duplicate dmi_dd_attr_info objects
Checking for duplicate dmi_dd_type_info objects
Checking for any dmi_dd_attr_info objects that are
missing an entry in dmi_dd_common_info_s
Checking for any dmi_dd_type_info objects that are
missing an entry in dmi_dd_common_info_s
Checking for any dmi_dd_attr_info objects that are
```

```
missing an entry in dmi_dd_attr_info_s
Checking for any dmi_dd_type_info objects that are
missing an entry in dmi_dd_type_info_s

#####
##
##
## CONSISTENCY_CHECK: Lifecycles
##
##      Start Time: 09-10-2002 10:16:11
##
#####

Checking for sysobjects that reference non_existent
policy objects
Checking for any policy objects that reference
non-existent types in included_type
Checking for any policy objects with missing
dm_sysobject_s entry
Checking for any policy objects with missing
dm_sysobject_r entries
Checking for policy objects with missing dm_policy_r
entries
Checking for policy objects with missing dm_policy_s
entry

#####
##
##
## CONSISTENCY_CHECK: FullText
##
##      Start Time: 09-10-2002 10:16:11
##
#####

Checking for tdk index objects that point to
non-existent fulltext index objects
Checking for any tdk collect objects that point to
non-existent tdk index objects
Checking for any fulltext index objects that point
to non-existent tdk index objects
Checking for any tdk index objects that point to
non-existent tdk collect objects
Checking for any non-orphaned dmr_content objects
that point to types that don't exist
Checking for any non-orphaned dmr_content objects
that point to non-existent formats
Checking for any dmr_content objects that point to
a non-existent fulltext index
Checking for any fulltext index attributes that are
no longer in dm_type

#####
##
##
## CONSISTENCY_CHECK: Indices
##
##      Start Time: 09-10-2002 10:16:11
##
```

```

#####
Checking for dmi_index objects that reference
  non-existent types
Checking for types with non-existent dmi_index
object for <type>_s table
Checking for types with non-existent dmi_index
object for <type>_r table
Checking for index objects with invalid attribute
  positions

#####
##
##
## CONSISTENCY_CHECK: Methods
##
##      Start Time: 09-10-2002 10:16:11
##
#####

Checking for java dm_method objects that reference
  jview

Consistency Checker completed successfully
Total number of inconsistencies found: 5
Disconnected from the server.

```

Content Replication

The Content Replication tool automates content replication between the component storage areas of a distributed storage area. The tool uses dump and load operations, but unlike manual dump and load operations, only requires enough temporary disk space to transfer the largest individual content file to be replicated. (A manual dump and load requires enough temporary space to hold a dump file containing all content files to be replicated.)

By default, the tool processes the content files in batches. It retrieves up to 500 content files (the default batch size) and releases resources in the source database before replicating the files. You can adjust the size of the batches by setting the `-batch_size` argument. Each execution of the job may process multiple batches, depending on the number of content files to be replicated and the batch size.

If the `-batch_size` argument is set to 0, the DQL hint `FETCH_ALL_RESULTS 0` is used in the query. All files to be replicated are cached in the Content Server's memory and transferred individually. Set `-batch_size` to 0 only if you have a very large amount of memory available.

If the `-batch_size` argument is set to 1, the `FETCH_ALL_RESULTS` hint is not used and query results are not cached.

If the `-batch_size` argument is set to any value greater than 1 and content transfer operations fail for the whole batch, the job exits and displays an error message.

The job uses a login ticket to connect to each source server. If you include the `-source_servers` argument, the job connects only to the servers in the list. If you do not include that argument, the job attempts to connect to each server in the repository.

Note: The clocks on the host machines of the source servers must be using UTC time and must be synchronized with the host machine on which the job runs. The login ticket for the job is valid for 5 minutes. If the clocks are not synchronized or the machines are using times set to different time zones, the source server to which the job is connecting may determine that the ticket has timed out and the job will fail.

A content replication job looks for all content not locally present, gets the files while connected to other sites, and performs an `IMPORT_REPLICA` for each content file in need of replication. The job generates a report that lists each object replicated. The report is saved to the repository in `/System/Sysadmin/Reports/ContentReplication`.

Note: If the report was run against the content at a remote distributed site, the report name will have the site's server config name appended. For example, if London is a remote site, its report would be found in `/System/Sysadmin/Reports/ContentReplicationLondon`.

Installing the tool suite at a site creates a content replication job for the installation site. In a distributed environment, the job's argument values for the remote sites are based on those of the Content Replication job for the primary site, but the job name and target server will be unique for each site. The job name has the format:

```
dm_ContentReplicationserverconfig.object_name
```

The job's `target_server` attribute identifies the local server performing the replication using the format `repository.serverconfig@hostname`.

The `ContentReplication` job is inactive by default.

Arguments

The Content Replication tool has the following arguments, described in [Table 12-5](#), page 478.

Table 12-5. Content Replication arguments

Argument	Datatype	Default	Description
<code>-window_interval</code>	integer	120	Defines window in which the tool can run. Value is interpreted in minutes.

Argument	Datatype	Default	Description
-queueperson	string(32)	-	User who receives email and inbox notifications from the tool. The default is the user specified in the operator_name attribute of the server config object.
-batch_size	integer	500	Number of content files to process in each batch.
-custom_predicate	string	-	Qualification applied to the content to be replicated. Enter what would normally appear after WHERE in a DQL qualification (For example, FOLDER ('/xyz', descend)
-source_servers	string	-	Comma-separated list of Content Servers to which to connect. Use the names of the servers' server config objects. The argument accepts a maximum of 255 characters. The specified names are recorded in the job's method_arguments attribute. If this argument is not included, the job attempts to connect to all other servers in the repository.

Report sample

Here is a sample of a ContentReplication report.

```
ContentReplication Report For repository wagnerdb
As Of 3/16/97 4:58:34 PM
Making lists of distributed components that are
local and far
Far Store: StoreC
Near Store: StoreE
Getting the source user for connecting to
other sites...
```

```
Getting the source password for connecting
to other sites...
Now connected to WagnerA
  Replicated 1 KB, format text for document
DBWarning
  Replicated 7 KB, format text for document
StateOfDocbase
  Replicated 14 KB, format text for document
LogPurge
  Replicated 2 KB, format text for document
ContentReplication
  Replicated 3 KB, format text for document
ContentWarning
  Replicated 5 KB, format text for document
DMClean
  Replicated 4 KB, format text for document
DMFilescan
  Replicated KB, format text for document
Disconnected from WagnerA
Report End 3/16/97 4:58:53 PM
```

Content Warning

The Content Warning tool notifies you when disks that you use for content storage approach a user-defined capacity. The notification is sent to the repository Inbox of the queueperson and as an email message. The tool also generates a report that is stored in the Reports folder under the Sysadmin folder in the System cabinet.

The tool determines where the repository is storing its content and then uses operating system commands to determine whether these disks are reaching the specified threshold. When the disk space used meets or exceeds the value in the tool's `percent_full` argument, a notification is sent to the specified queueperson and a report is generated and saved to the Docbase in `/System/Sysadmin/Reports/ContentWarning`.

Note: If the tool was run against the content at a remote distributed site, the report name will have the site's server config name appended. For example, if London is a remote site, its report would be found in `/System/Sysadmin/Reports/ContentWarningLondon`.

The Content Warning tool is installed in the active state by default.

Arguments

The Content Warning tool has three arguments, described in [Table 12-6, page 481](#).

Table 12-6. Content Warning arguments

Argument	Datatype	Default	Description
-percent_full	integer	85	Percent-full threshold at which a message is sent.
-queueperson	string(32)	-	User who receives email and inbox notifications from the tool. The default is the user specified in the operator_name attribute of the server config object.
-window_interval	integer	720	Execution window for the tool, expressed in minutes. Refer to The window interval, page 462 for a complete description.

Report sample

Here is a sample of a ContentWarning report.

```
Object: test1_file_store
Type : dm_filestore
Path : /export/nfs2-4/dmadmin/data/test1/
test1_storage_location
  Total Disk   Total Used   Total Free   Percent Used
  1,952,573    1,620,019    137,304      93
DocBasic Total Free:                1,124,794
Content File (Document) Space Utilization
In test1_file_store
-Active          2,485,090
-Deleted          81,397
-Total           2,566,487

Object: test1_file_store2
Type : dm_filestore
Path : /export/nfs2-1/dmadmin/data/test1/storage_02
Total Disk   Total Used   Total Free   Percent Used
  1,952,573    1,113,666    643,657      67
DocBasic Total Free:                977,871
Content File (Document) Space Utilization
```

```

In test1_file_store2
-Active          733,532
-Deleted         3,821
-Total          737,352

Object:  support_file_store
Type :   dm_filestore
Path :   /export/nfs2-1/dmadmin/data/test1/docs_from_test2

Total Disk   Total Used   Total Free   Percent Used
 1,952,573   1,113,666   643,657     67

DocBasic Total Free:          977,871

Content File (Document) Space Utilization In
test2_file_store
-Active          863
-Deleted
-Total          863

```

Create Full-Text Events

The Create Full-Text Events tool (`dm_FTCreateEvents`) may be used in two ways:

- To complete an upgrade by causing any objects missed by the pre-upgrade indexing operations to be indexed

The job generates events for each indexable object added to a repository between the time a new 5.3 or later full-text index is created for a 5.2.5 repository and when the repository is upgraded to 5.3

For example, a copy of a 5.2.5 repository can be used to create a new 5.3 index. Depending on when the production repository is upgraded, new indexable objects may be created in the production repository after the new 5.3 index is created. When the production repository is upgraded to 5.3 and begins to use the new index, the repository contains objects that are not yet indexed. Running `dm_FTtCreateEvents` generates events for the new objects. An index agent running in normal mode uses the events to submit the objects for indexing.

This is the default behavior of the job.

- To generate the events required to reindex an entire 5.3 SP1 or later repository

The `-full_reindex` argument must be set to `TRUE` to generate the required events. Reindexing the repository does not require deleting the existing index.

The tool itself does not update the index. The tool can optionally generate a list of object IDs of objects that must be indexed, rather than generating events for those objects. The list is used to submit objects to the index agent in file mode for indexing. The *Content Server Full-Text Indexing Installation Manual* contains instructions for using the index agent's file mode.

The first time the job runs in its default mode, the job determines the last object indexed by an index agent running in migration mode and the date on which that object was indexed. The job searches for objects modified after that date and before the job runs for the first time and generates events for those objects. On its subsequent iterations, the job searches for objects modified after the end of the last iteration and before the beginning of the current iteration.

Before the job is run in a 5.3 SP1 or later repository with `full_reindex` set to `TRUE`, you must create a high-water-mark queue item (`dmi_queue_item`) manually using the API and specify the `r_object_id` of the queue item as the `-high_water_mark_id` argument of the Create Full-Text Events tool. Using IAPI or the API window in Documentum Administrator, the commands to create the queue item are:

```
create,c,dmi_queue_item
save,c,l
```

Use the object ID generated by the create command in the `-high_water_mark_id` argument of the tool.

By default, the tool is installed in the active state to run daily at 11 p.m. The tool processes new objects in batches of 50,000. If all objects are not processed in one run, the tool continues executing each day at 11 p.m. When it finds no more objects to process, it sets itself to inactive. When the `-full_reindex` argument is set to `TRUE`, the events are created in reverse chronological order by `r_modify_date` and therefore the most recently-modified or created objects are indexed first.

Create Full-Text Events generates a status report that is saved in the repository in `/System/Sysadmin/Reports/FTCreateEvents`.

Arguments

The Create Full-Text Events tool has nine arguments, described in [Table 12-7, page 484](#). When the job is run in the default mode, all arguments are provided by the job itself. When the job is run in full-reindex mode, you must set `-full_reindex` to `TRUE` and provide the `r_object_id` of the manually-created queue item as the value of `-high_water_mark_id`.

Table 12-7. Create Full-Text Events arguments

Argument	Datatype	Default	Description
-max_events_per_run	integer	50000	The maximum number of events generated each time the job runs. If max_events_per_run is not set or is set to zero, the job creates events for all objects changed or created between min_date and max_date.
-high_water_mark_id		-	Object ID of the queue item to use for obtaining the last date for which queue items were created. If it is not specified, the job queries for the most recently created dmi_queue_item in which the value of item_name is "Full-text re-index high water mark" and the value of task_state is "done." In full reindex mode, you must provide the r_object_id of the manually-created high-water-mark queue item.

Argument	Datatype	Default	Description
-min_date	Date	When not in full reindex mode, the value of the <code>date_sent</code> attribute of the migration-mode queue item for indexing.	The earliest date on which objects were modified for which the job creates events.
-max_date	Date	When not in full reindex mode, the date on which the job runs for the first time.	The most recent date on which objects were modified for which the job creates events.
-file	string	-	When submitted with a filename, the object IDs of objects that require events are written to a file. The syntax is: <code>-file full_path_of_file</code>
-full_reindex	Boolean	FALSE	When set to false, the job generates events for each indexable object added to a repository between the time a new 5.3 full-text index is created for a 5.2.5 repository and when the repository is upgraded. This is the behavior of the job when it is installed. When set to TRUE, events are generated for all indexable

Argument	Datatype	Default	Description
-current_only	Boolean	FALSE	objects in reverse chronological order by the value of the r_modify_date. Use when -full_reindex is set to TRUE. When set to TRUE, new events are generated only for the CURRENT version of each indexable object.
-queueperson	string(32)	-	User who receives email and inbox notifications from the tool. The default is the user specified in the operator_name attribute of the server config object.
-window_interval	integer	12000	Execution window for the tool, expressed in minutes. Refer to The window interval, page 462 for a complete description.

Report sample

Here is a sample of a Create full-text events report.

```

2005/07/19 14:07:41:552 -----
2005/07/19 14:07:42:395 Located High Water Mark dmi_queue_item '1b0014dc8000210e'
      as the most recent one that was marked complete.
2005/07/19 14:07:42:395 min date from hwm: '07/18/2005 18:26:27'.
2005/07/19 14:07:42:395 High Water Mark dmi_queue_item has no router_id set,
      hence it's a brand new iteration of this job.
```

```

2005/07/19 14:07:42:395 Updating High Water Mark '1b0014dc8000210e',
      setting actual_start_date to '07/19/2005 14:07:42'.
2005/07/19 14:07:42:474 Query begin: select r_object_id, r_modify_date from
dm_sysobject (all) where r_modify_date >= DATE('07/18/2005 18:26:27',
'mm/dd/yyyy hh:mi:ss') and r_modify_date < DATE('07/19/2005 14:07:42',
'mm/dd/yyyy hh:mi:ss') and r_object_id > '000000000000000000'
order by r_object_id
2005/07/19 14:07:42:505 Query end
2005/07/19 14:07:42:505 Query result: 080014dc800002c2 07/19/2005 14:07:41
2005/07/19 14:07:42:708 Query result: 090014dc80001906 07/19/2005 14:03:43
2005/07/19 14:07:42:755 Query result: 090014dc80001908 07/19/2005 14:03:43
2005/07/19 14:07:42:786 Created 3 events or dmi_queue_item objects
      (Batch Size used was 50000).
2005/07/19 14:07:42:786 Done creating events.
      Updating the High Water Mark '1b0014dc8000210e',
by setting the router_id to '0000000000000000'.
2005/07/19 14:07:42:895 -----
Report End 2005/07/19 14:07:42

```

Data Dictionary Publisher

The Data Dictionary Publisher tool publishes the data dictionary information. The data dictionary is information about object types and attributes stored in internal objects by Content Server and made available to client applications through the publishing operation. Publishing the information creates dd type info and dd attr info objects. These are persistent objects whose attributes store the data dictionary information. Client applications that use the data dictionary information can reference or query these objects and their attributes. (For more information about the data dictionary, refer to [The data dictionary, page 64](#), in *Content Server Fundamentals*.)

Data Dictionary Publisher generates a status report that is saved in the repository in `/System/Sysadmin/Reports/DataDictionaryPublisher`.

Arguments

The Data Dictionary Publisher job has one argument, described in [Table 12–8, page 488](#).

Table 12-8. Data Dictionary Publisher argument

Argument	Datatype	Default	Description
-window_interval	integer	720	Execution window for the tool, expressed in minutes. Refer to The window interval, page 462 for a complete description.

Report sample

```

Connected To sqlntX.sqlntX
Job Log for System Administration Tool
DataDictionaryPublisher
-----

This job log consists of three distinct parts:
1) All print statements from the execution of the job
2) The report for the tool which is saved as a
   separate document in the Docbase in
   /System/Sysadmin/Reports.
3) The trace file results from the trace API,
   if the job's trace level is > 0.

Note: The report and trace file are also maintained
      under the Documentum log location in:
$DOCUMENTUM/dba/log/<docbase hex id>/sysadmin
They are overwritten each time the job executes.

Start of log:
-----
DataDictionaryPublisher Tool Completed at
11/8/2000 12:15:25. Total duration was 0 minutes.
Calling SetJobStatus function...

--- Start
c:\Documentum\dba\log\000145df\sysadmin\
DataDictionaryPublisherDoc.txt report output ----
DataDictionaryPublisher Report For DocBase sqlntX
As Of 11/8/2000 12:15:24
DataDictionaryPublisher utility syntax:
  apply,c,NULL,EXECUTE_DATA_DICTIONARY_LOG
Executing DataDictionaryPublisher...
Report End 11/8/2000 12:15:25

--- End
c:\Documentum\dba\log\000145df\sysadmin\
DataDictionaryPublisherDoc.txt report output ---

```

Database Space Warning

The Database Space Warning tool scans the RDBMS to determine:

- How full the tablespace (Oracle) or device (Sybase) is
- Whether any tables are fragmented beyond a user-specified limit
- Whether the expected number of Documentum indexes are present

The tool also recreates any indexes that are identified by `dmi_index` objects but not found in the database.

Note: The Database Space Warning Tool is not needed, and therefore not installed, for installations running against MS SQL Server or DB2.

If the tool finds that the space has reached the limit specified in the tool's `percent_full` argument, it sends a notification to the user specified in `queueperson`. When it sends a notification, it also includes a message about any RDBMS tables that are fragmented beyond the limits specified in the `max_extents` argument and a message regarding indexes, if it does not find the expected number in the RDBMS. The notifications are sent to the user's repository Inbox and through email.

In addition to these notifications, the tool generates a status report that is saved in the repository in `/System/Sysadmin/Reports/DBWarning`.

The Database Space Warning tool is installed in the active state.

For Sybase, you must set the `ddl in tran` database option to `TRUE` to run this job. The `isql` syntax is:

```
sp_dboption dbname, "ddl in tran", true
```

where *dbname* is the name of the database for your repository.

Arguments

The Database Space Warning tool has four arguments, described in [Table 12-9, page 489](#).

Table 12-9. Database Space Warning arguments

Argument	Datatype	Default	Description
<code>-percent_full</code>	integer	85	Percent-full threshold at which a message is sent.
<code>-max_extents</code>	integer	50	The number of extents that an RDBMS table may have before being reported as fragmented.

Argument	Datatype	Default	Description
-queueperson	string(32)	-	User who receives email and inbox notifications from the tool. The default is the user specified in the operator_name attribute of the server config object.
-window_interval	integer	720	Execution window for the tool, expressed in minutes. Refer to The window interval, page 462 for a complete description.

Report sample

Here is a sample of a Database Space Warning report. It shows the total number of blocks allocated for the database, how many are currently used for tables and indexes, the percentage used of the total allocated, and the number of free blocks. It also lists the number of fragments for all tables and indexes with more than max_extents fragments and lists the number of Documentum indexes in the repository.

```
Database Block Allocation
Table  Index  TotalUsed  Free      Total  %Used/Total
620    647      1,267     81,929   83,196  2
```

```
DBMS Tables With Multiple Extents
# of Segs  Type      Name
6          TABLE   DM_TYPE_R
5          INDEX    DMINDEX_1F00096380000009
4          TABLE   DM_SYSOBJECT_S
3          TABLE   DMI_OBJECT_TYPE
3          INDEX    DMI_OBJ_TYPE_INDEX
3          INDEX    DMI_OBJ_ID_INDEX
3          TABLE   DM_FORMAT_S
3          TABLE   DM_SYSOBJECT_R
```

Inbox and email message samples

Here is a sample Inbox message sent by the Database Space Warning tool:

```
Take a look at your DBMS tablespace--it's 90% full!
You have 8 fragmented tables in your DBMS instance
--you may want to correct this!
You are missing some Documentum indexes-contact Support!
```

Here is the corresponding email message:

```
Return-Path: <dmadmin@bigcat>
X-UIDL: 827349620.001
Date: Wed, 20 Mar 1996 11:18:54 -0800
From: dmadmin@bigcat (Documentum 2.0)
To: stevex@tiger
Subject: Event FragggedTables has occurred
on DBWarning.Doc
(090000018006ee33) by dm20
```

```
DOCBASE:          test1
EVENT:            FragggedTables
NAME:             DBWarning.Doc
SENT BY:          dmadmin
TASK NAME:        event
```

```
MESSAGE:
You have 18 fragmented tables in your DBMS instance
--you may want to correct this!
```

Dm_LDAPSynchronization

The dm_LDAPSynchronization tool finds the changes in the user and group information in an LDAP-compliant directory server that have occurred since the last execution of the tool and propagates those changes to the repository. The tool sets default values for user_global_unique_id and group_global_unique_id. If necessary, the tool creates default folders and groups for new users. If there are mapped user attributes, those are also set.

Exactly what operations the tool can perform depends on what kind of directory server is in use. If you are using Netscape iPlanet Directory Server, Oracle Intranet Directory Server, or MS Active Directory on a Windows platform, the tool can:

- Import new users and groups in the directory server into the repository
- Rename users in the repository if their names changed in the directory server
- Rename groups in the repository if their names changed in the directory server
- Inactivate users in the repository that if they were deleted from the directory server.

When using iPlanet, you must enable the changelog feature to use the renaming and inactivation operations. Instructions for enabling the changelog feature are found in the vendor's iPlanet Administration Guide.

The renaming and inactivation operations are not supported on MS Active Directory on UNIX platforms.

The behavior of the tool is determined by the attribute settings of the dm_ldap_config object. The tool has four arguments that you can use to override the attribute settings controlling which operations the tool performs. These are listed in [Table 12-10, page 492](#).

The dm_LDAPSynchronization tool requires the Java method server. Ensure that the Java method server in your Content Server installation is running.

The dm_LDAPsynchronization tool generates a report that is saved in the repository in /System/Sysadmin/Reports/LDAPsynchronization.

The tool is installed in the inactive state. After it is activated, it is executed once a day at 4 a.m. by default. Before you set it to the active state, you must define the ldap_config object for the repository. For instructions about defining the set-up values, refer to [Defining the set-up values, page 359](#).

Arguments

The dm_LDAPsynchronization job has eight arguments, described in [Table 12-10, page 492](#).

Table 12-10. dm_LDAPsynchronization arguments

Argument	Datatype	Default	Description
-deactivate_user_option	Boolean	FALSE	Set to TRUE, directs the job to inactivate users in the repository that have been deleted from the directory server. Setting this overrides the deactivate_user_option attribute in the ldap config object.
-full_sync	Boolean	FALSE	Set to TRUE, this directs the job to retrieve all entries from the LDAP directory that satisfy the search criteria. FALSE causes the job to import into the repository only new or updated LDAP entries.
-import_mode	string(7)	all	Controls whether the job imports users, groups, or both into the repository. Valid values are: users, groups, and all. Setting this overrides the import_mode attribute in the ldap config object.

Argument	Datatype	Default	Description
-queueperson	string(32)	-	User who receives email and inbox notifications from the tool. The default is the user specified in the operator_name attribute of the server config object.
-rename_group_option	Boolean	FALSE	Set to TRUE, directs the job to rename groups in the repository if their names have changed in the directory server. Setting this overrides the rename_group_option attribute in the ldap config object.
-rename_user_option	Boolean	FALSE	Set to TRUE, directs the job to rename users in the repository if their names have changed in the directory server. Setting this overrides the rename_user_option attribute in the ldap config object.
-source_directory		dm_all_directories	Controls which LDAP servers are synchronized when the job runs. If not set, all LDAP servers associated with the server config object are synchronized. If set to particular LDAP servers, only those servers are synchronized.
-window_interval	integer	1440	Execution window for the tool. Refer to The window interval, page 462 for a complete description.

Executing dm_LDAPsynchronization manually

You can execute the dm_LDAPsynchronization tool manually from the command line. The syntax is

```
java com.documentum.ldap.LDAPSync -doibase_name repositoryname  
-user_name superuser_login -method_trace_level integer -full_sync true
```

where *repositoryname* is the name of the repository, *superuser_login* is the login for a Superuser, and *integer* is the required trace level for the method.

Dmclean

The Dmclean tool automates the dmclean utility. The utility scans the repository for orphaned content objects, ACLs, annotations (dm_note objects), and aborted workflows. The utility also scans for the workflow templates created by the SendToDistributionList command (a Documentum Desktop command that routes a document to multiple users concurrently) and left in the repository after the workflow completed. The utility generates an API script to remove these orphans. (For detailed information about the dmclean utility, refer to [Chapter 7, Content Management](#).) The Dmclean tool performs dmclean's operations and (optionally) runs the generated script.

When the agent exec program invokes the script, the tool generates a report showing what content objects, content files, ACLs, notes and workflow templates would be removed upon execution of the generated script. The status report is saved in /System/Sysadmin/Reports/DMClean.

Note: If the tool was run against the content at a remote distributed site, the report name will have the site's server config name appended. For example, if London is a remote site, its report would be found in /System/Sysadmin/Reports/DMCleanLondon.

Whether the generated script runs is controlled by the tool's clean_now argument. This argument is set to TRUE by default. If you set it to FALSE, the script is not run, and you will have to run it manually to remove the orphan objects. The script is stored in %DOCUMENTUM\dba\log\hexrepositoryid\sysadmin (\$DOCUMENTUM/dba/log/hexrepositoryid/sysadmin).

The Dmclean tool is installed in the inactive state.

Arguments

[Table 12–11, page 495](#), describes the arguments accepted by Dmclean.

Table 12-11. Dmclean arguments

Argument	Datatype	Default	Description
-clean_content	Boolean	TRUE	Controls whether the tool searches for orphaned content objects. Set to FALSE if you do not want to include content objects in the dmclean operation.
-clean_note	Boolean	TRUE	Controls whether the tool searches for orphaned note objects (annotations). Set to FALSE if you do not want to include notes in the dmclean operation. Note: Even if the setting is TRUE, the tool never removes a note whose object ID is recorded in an audit trail entry (a dm_audittrail object).
-clean_acl	Boolean	TRUE	Controls whether the tool searches for orphaned ACLs. Set to FALSE if you do not want to include ACLs in the dmclean operation.
-clean_now	Boolean	TRUE	Controls whether the tool removes the orphaned objects. The dmclean utility generates an API script to clean the repository. Setting clean_now to TRUE automatically executes the script as part of the job.
-clean_wf_template	Boolean	TRUE	Controls whether the tool removes old workflow templates created when users executed the SendToDistributionList menu command from a Documentum client menu.

Argument	Datatype	Default	Description
-clean_aborted_wf	Boolean	FALSE	Controls whether the tool removes aborted workflows and all the workflows' associated runtime objects (packages, work items, and so forth) from the repository.
-clean_castore	Boolean	FALSE	Controls whether the tool includes orphaned content with expired retention dates in ca store storage areas in the operation. T means that expired, orphaned content in ca store storage areas is included in the operation.
-queueperson	string(32)	-	User who receives email and inbox notifications from the tool. The default is the user specified in the operator_name attribute of the server config object.
-window_interval	integer	120	Execution window for the tool. Refer to The window interval, page 462 for a complete description.

Guidelines

If you are using distributed content, dmclean requires the default storage area for dm_sysobjects to be the distributed store.

How often you run Dmclean will depend on

- Your business rules
- The size of the repository
- The amount of storage capacity

Typically, including notes and ACLs in the operation noticeably increases the execution time of the tool, so you may want to reset these arguments to FALSE on some runs.

If you prefer to review what and how much will be deleted before executing the API script, set the `clean_now` argument to `FALSE`. (You will have to execute the script manually after inspection.)

Report sample

Here is a sample of a Dmclean report:

```
DMClean Report For DocBase testdoc
As Of 5/14/2002 11:58:46

Arguments for the dmclean method:
Unused annotation objects will be cleaned up...
Unused internal ACL objects will be cleaned up...
Unused SendToDistributionList workflow template
objects will be cleaned up...
Orphaned content objects will be cleaned up...
Generated DMClean script will be executed...

The trace level is set to 0...
DMClean utility syntax: apply,c,NULL,DO_METHOD,
METHOD,S,dmclean

Executing DMClean...
All Clean contents were successfully removed.
Generated script from the DMClean method:
----- Start
C:\Documentum\dba\log\000003e8\sysadmin\
080003e8800005d6.bat
output -----
# Opening document base testdoc...
#   Total shared memory size used: 1554112 bytes
#   Making /System cabinet.
#   /System cabinet exists.
#   Making /Temp cabinet.
#   /Temp cabinet exists.
#   Making /System/Methods folder.
#   /System/Methods folder exists.
#   Making /System/FileSystem folder.
#   /System/FileSystem folder exists.
#   Making /System/DataDictionary folder.
#   /System/DataDictionary folder exists.
#   Making /System/Procedures folder.
#   /System/Procedures folder exists.
#   Making /System/Procedures/Actions folder.
#   /System/Procedures/Actions folder exists.
#   Making /System/Distributed References folder.
#   /System/Distributed References folder exists.
#   Making /System/Distributed References/Links
folder.
#   /System/Distributed References/Links folder
exists.
#   Making /System/Distributed References/Checkout
folder.
#   /System/Distributed References/Checkout folder
```

```
exists.
# Making /System/Distributed References/Assemblies
  folder.
# /System/Distributed References/Assemblies folder
exists.
# Making /System/Distributed References/Workflow
  folder.
# /System/Distributed References/Workflow folder
exists.
# Making /System/Distributed References/VDM folder.
# /System/Distributed References/VDM folder exists.
# Making docbase config object.
# Making server config object.
#
# Documentum, Inc.
#
# dmclean cleans up orphan content, annotation,
# internal ACL, and unused SendToDistributionList
# workflow template objects. Instead of immediately
# destroying the orphan content objects, dmclean
# generates an API script, which can
# be used for verification before cleanup actually
# happens.
# This is done in this manner because deleted content
# objects by mistake are difficult to recover.
# dmclean, however, cleans up all unused annotations
# and internal ACLs.
# To remove orphan content objects after verification,
# do the following in iapi:
#
# % iapi <DOCBASE> -U<USER> -P<PWD>
# API> @<SCRIPT_NAME>
# API> quit
#
# Starting to clean up unused content objects...
# Content object 060003e880002100 has parent
# count of zero.
apply,c,060003e880002100,DESTROY_CONTENT
getmessage,c
close,c,q0
# Content object 060003e880002101 has parent
# count of zero.
apply,c,060003e880002101,DESTROY_CONTENT
getmessage,c
close,c,q0
# Content object 060003e880002105 has parent
# count of zero.
apply,c,060003e880002105,DESTROY_CONTENT
getmessage,c
close,c,q0
# Content object 060003e88000210c has parent
# count of zero.
apply,c,060003e88000210c,DESTROY_CONTENT
getmessage,c
close,c,q0
# Content object 060003e880002114 has parent
# count of zero.
```

```

apply,c,060003e880002114,DESTROY_CONTENT
getmessage,c
close,c,q0
# Content object 060003e880002119 has parent
# count of zero.
apply,c,060003e880002119,DESTROY_CONTENT
getmessage,c
close,c,q0
# Count of objects with zero parent count was: 6
# Content cleanup complete.
# Starting to clean up unused subcontent objects...
# SubContent cleanup complete.
# Starting to Remove unused annotations...
# Total number of annotations removed: 0
# Starting to clean up unused internal ACLs...
# Total number of ACLs removed: 0
# Internal ACL clean up complete.
# Starting to Remove unused SendToDistributionList
# workflow templates...
# Total number of SendToDistributionList workflow
# templates removed: 0
----- End
C:\Documentum\dba\log\000003e8\sysadmin\
080003e8800005d6.bat output -----
Destroying DMClean script with ID 090003e880002907...
Report End 5/14/2002 11:59:16

```

Dmfilesan

The Dmfilesan tool automates the dmfilesan utility. This utility scans a specific storage area or all storage areas for any content files that do not have associated content objects and generates a script to remove any that it finds. The tool executes the generated script by default, but you can override the default with an argument. (For detailed information about the dmfilesan utility, refer to [Using dmfilesan, page 273](#).)

Dmfilesan also generates a status report that lists the files it has removed. The report is saved in the repository in `/System/Sysadmin/Reports/DMFilesan`.

Note: If the tool was run against the content at a remote distributed site, the report name will have the site's server config name appended. For example, if London is a remote site, its report would be found in `/System/Sysadmin/Reports/DMFilesanLondon`.

Dmfilesan is installed in the inactive state.

Arguments

The Dmfilescan tool are described in [Table 12-12, page 500](#). Refer to the description of the dmfilescan utility in [Using dmfilescan, page 273](#), for instructions on specifying values for the `-from` and `-to` arguments.

Table 12-12. Dmfilescan arguments

Argument	Datatype	Default	Description
<code>-s <i>storage_name</i></code>	string	-	Specifies a target storage area. If this argument is not included, all storage areas are scanned.
<code>-from <i>directory_path</i></code>	string	-	Starting subdirectory for the scan operation. Refer to Identifying the subdirectories of the scanned storage areas, page 275 for information about using this argument.
<code>-to <i>directory_path</i></code>	string	-	Ending subdirectory for the scan operation. Refer to Identifying the subdirectories of the scanned storage areas, page 275 for information about using this argument.
<code>-scan_now</code>	Boolean	TRUE	Controls whether the generated script is executed. TRUE (the default) executes the generated script. Set this to FALSE if you want to execute the script manually.
<code>-queueperson</code>	string(32)	-	User who receives email and inbox notifications from the tool. The default is the user specified in the <code>operator_name</code> attribute of the server config object.

Argument	Datatype	Default	Description
-window_interval	integer	120	Execution window for the tool. Refer to The window interval, page 462 for a complete description.
-force_delete	Boolean	FALSE	Controls whether orphan files created within 24 hours of the job's execution are deleted. Refer to the Guidelines for details.
-no_index_creation	Boolean	FALSE	Controls whether dmfilesan creates and destroys the indexes on dmr_content.data_ticket and dmr_content.other ticket or assumes they exist. T (TRUE) means that the utility assumes that the indexes exist prior to the start of the utility. F (FALSE) means the utility will create these indexes on startup and destroy them at the finish. Refer to Using the -no_index_creation argument, page 276 for details of use.

Guidelines

Typically, if you run the Dmclean tool regularly, it is not necessary to run the Dmfilesan tool more than once a year. By default, the tool removes all orphaned files from the specified directory or directories that are older than 24 hours. If you wish to remove orphaned files younger than 24 hours, you can set the -force_delete flag to T (TRUE). However, this flag is intended for use only when you must remove younger files to clear disk space or to remove temporary dump files created on the target that were not removed automatically. If you execute Dmfilesan with -force_delete set to T, make sure that there are no other processes or sessions creating objects in the repository at the time the job executes.

If you are using distributed content, dmfilesan requires the default storage area for dm_sysobjects to be the distributed store.

Report sample

The following is a sample of a Dmfilescan report.

```
DMFilescan Report For DocBase boston2
As Of 9/17/96 11:08:54 AM
  Generated DMFilescan script will be executed...
  The trace level is set to 5...
DMFilescan utility syntax: apply,c,NULL,DO_METHOD,
METHOD,S,dmfilescan
Executing DMFilescan...
Executing DMFilescan script...
sh /u106/dm/dmadmin/dba/log/00000962/sysadmin/
0900096280012800.bat
>/u106/dm/dmadmin/dba/log/00000962/sysadmin/
0900096280012800.txt
Generated script from the DMFilescan method:
----- Start
/u106/dm/dmadmin/dba/log/00000962/sysadmin/
0900096280012800.bat output
-----
#!/bin/sh -x
#
# Documentum, Inc.
#
# This script is generated by dmfilescan for later
# verification and/or clean-up. This script is in
# trace mode by default. To turn off the trace mode,
# remove the '-x' in the first line.
#
# To see if there are any content objects referencing
# a file reported below, use the following query
# (executed in idql):
#
# % idql <docbase> -U<user> -P<pwd>
# 1> select r_object_id from dmr_content
# 2> where storage_id = '<storage_id>' and data_ticket =
# <data_ticket>
# 3> go
#
# If there are no rows returned, then this is an
# orphan file.
#
# Opening document base boston2...
#   Making distributed object_id map.
#   Making /System cabinet.
#   /System cabinet exists.
#   Making /Temp cabinet.
#   /Temp cabinet exists.
#   Making /System/Methods folder.
#   /System/Methods folder exists.
#   Making /System/FileSystem folder.
#   /System/FileSystem folder exists.
#   Making docbase config object.
#   Making server config object.
# Document base boston2 opened. Starting filescan...
```

```
# Building indexes for content lookups ...
# Checking store filestore_01...
# Checking store replica_filestore_01...
# Checking store replicate_temp_store...
# Reading directory '/u127/dm/data/boston2/
content_storage_01/00000962'
# Reading directory '/u127/dm/data/boston2/
content_storage_01/00000962/80'
# Reading directory '/u127/dm/data/boston2/
content_storage_01/00000962/80/00'
# Reading directory '/u127/dm/data/boston2/
content_storage_01/00000962/80/00/01'
# Reading directory '/u127/dm/data/boston2/
content_storage_01/00000962/80/00/02'
# Reading directory '/u127/dm/data/boston2/
content_storage_01/00000962/80/00/03'
# Reading directory '/u127/dm/data/boston2/
content_storage_01/00000962/80/00/04'
# Reading directory '/u127/dm/data/boston2/
content_storage_01/00000962/80/00/05'
# Reading directory '/u127/dm/data/boston2/
content_storage_01/00000962/80/00/06'
# Reading directory '/u127/dm/data/boston2/
content_storage_01/00000962/80/00/07'
# Reading directory '/u127/dm/data/boston2/
content_storage_01/00000962/80/00/08'
# Reading directory '/u127/dm/data/boston2/
content_storage_01/00000962/80/00/09'
# Reading directory '/u127/dm/data/boston2/
content_storage_01/00000962/80/00/0a'
# Reading directory '/u127/dm/data/boston2/
content_storage_01/00000962/80/00/0b'
# Reading directory '/u127/dm/data/boston2/
content_storage_01/00000962/80/00/0c'
# Reading directory '/u127/dm/data/boston2/
content_storage_01/00000962/80/00/0d'
# Reading directory '/u127/dm/data/boston2/
content_storage_01/00000962/80/00/0e'
# Reading directory '/u127/dm/data/boston2/
content_storage_01/00000962/80/00/0f'
# Reading directory '/u127/dm/data/boston2/
content_storage_01/00000962/80/00/10'
# Reading directory '/u127/dm/data/boston2/
content_storage_01/00000962/80/00/11'
# Reading directory '/u127/dm/data/boston2/
content_storage_01/00000962/80/00/12'
# Reading directory '/u127/dm/data/boston2/
content_storage_01/00000962/80/00/13'
# Reading directory '/u127/dm/data/boston2/
replica_content_storage_01/00000962'
# Reading directory '/u127/dm/data/boston2/
replica_content_storage_01/00000962/80'
# Reading directory '/u127/dm/data/boston2/
replica_content_storage_01/00000962/80/00'
# Reading directory
'/u127/dm/data/boston2/replica_content_storage_01/
00000962/80/00/01'
```

```
# Reading directory
'/u127/dm/data/boston2/replica_content_storage_01/
00000962/80/00/02'
# Reading directory
'/u127/dm/data/boston2/replica_content_storage_01/
00000962/80/00/03'
# Reading directory
'/u127/dm/data/boston2/replica_content_storage_01/
00000962/80/00/09'
# Reading directory
'/u127/dm/data/boston2/replica_content_storage_01/
00000962/80/00/0a'
# Reading directory
'/u127/dm/data/boston2/replica_content_storage_01/
00000962/80/00/0c'
# Reading directory
'/u127/dm/data/boston2/replica_content_storage_01/
00000962/80/00/07'
# Reading directory '/u127/dm/data/boston2/
temp_replicate_store/00000962'
# Directory /u127/dm/data/boston2/temp_replicate_store/
00000962 is empty
# 0 orphan files were found
#
# Cleaning up content indexes ...
----- End /u116/dm/dmadmin/dba/log/00000962/sysadmin/
0900096280012800.bat output
-----
Destroying DMFilesScan result file with ID 0900096280012800...
Report End 9/17/96 11:09:54 AM
```

File Report

The File Report tool assists you in restoring deleted repository documents. It generates a report that lists all documents in the repository and their corresponding content files. Using that report in conjunction with a file system backup, you can restore the content file of a deleted document. (Refer to [Using a report to restore a document](#), page 506, for instructions on restoring a document.)

If a document must be recreated, these reports identify which files must be restored to rebuild the document. The system administrator matches lost documents to the file names so that the content files can be recovered. This feature is especially useful for restoring a single document (or a small set of documents) to a previous state, which cannot be done from database backups.

The File Report tool, as installed, runs a full report once a week against all file storage areas in the repository. It is possible to run incremental reports and reports that only examine a subset of the storage areas for the repository. For instructions on setting these up, refer to [Creating incremental or partial-repository reports](#), page 505.

Note: File Report only provides a mechanism for restoring the document content. The document metadata must be restored manually.

File Report saves the generated report to `/System/Sysadmin/Reports/FileReport`.

The File Report tool is installed as inactive.

Guidelines

Set up the File Report schedule on the same interval as the file system backups. For example, if nightly backups are done, also run File Report nightly and store the resulting report with the backups.

We recommend scheduling nightly incremental reports and generating full repository reports on a less frequent basis (weekly or bi-weekly).

If your repository is so large that creating full reports is not practical or generates cumbersome files, set up multiple jobs, each corresponding to a different storage area.

Usage notes

This section describes two procedures for using file reports:

- Creating new file report jobs to create incremental reports or reports for a subset of storage areas.
- Using file reports to recover a document

Creating incremental or partial-repository reports

A File Report job creates an incremental report if its `-incremental_report` argument is set to `TRUE`. Incremental reports only include documents that have changed since the last File Report was run.

If you include the `-storage_area` argument, the job generates a report on the documents in the specified storage area.

If you include the `-folder_name` argument, the job generates a report on documents in the specified folder.

Including both the `-storage_area` and `-folder_name` arguments generates a report on those documents in the specified folder that are also stored in the given storage area.

To create a job that generates incremental reports or only reports on some storage areas, copy an existing File Report job object and set its attributes and arguments as needed. Provide a new name for the copy that identifies it meaningfully.

[Chapter 2, Content Repositories](#), provides instructions for creating new jobs, and [Job, page 270](#), of the *EMC Documentum Object Reference Manual* contains a description of the `dm_job` object's attributes. You can create or copy a job using Documentum Administrator or the API.

Using a report to restore a document

To restore a document to the repository:

1. Find the last backup file report with the document listed in it.
2. Find the name(s) of the content file(s) which comprise the document.
3. Restore the named files from your file system backups to the original storage area.
This restores the content files to the storage area directory. This *does not* restore the documents to the repository. Until the files are imported back into the repository, they are treated as *orphaned* content files and will be removed by the next invocation of the `dm_clean` utility.

If you wish, you can move these files out of the storage area directories to a more appropriate work area in order to import them into the repository.

4. Use the restored content files to recreate the repository documents.
The best way to do this is to use a Documentum client to recreate the object metadata and then use an API session on the server machine to restore the content.

Restore the first primary content page using:

```
setfile,c,object_id,restored_content_file,format_type
```

If restoring a multiple page document, restore subsequent primary pages with:

```
setfile,c,object_id,restored_content_file,page_no
```

If you need to restore renditions of the document pages manually, use the `Addrendition` method. (Refer to [Addrendition, page 66](#), in the *Content Server API Reference Manual* for instructions about the syntax.)

You can restore documents that have only one content file using the client's `Import` function if the content files are directly accessible by the client. Because the content files restored from file system backups are written to the *server* storage areas, you must either directly access those directories from the client or copy the restored files to a network disk and import them from there.

If the document has multiple pages, use API methods to restore it.

Arguments

The File Report arguments are described in [Table 12–13, page 507](#).

Table 12-13. File Report arguments

Argument	Datatype	Default	Description
-folder_name <i>folder_path</i>	string	-	Identifies a folder path on which to run the report. May be used in conjunction with the -storage-area argument.
-incremental_report	Boolean	FALSE	When set to TRUE, the report is run incrementally. An incremental report only reports documents modified since the last time the job ran. (A full report is generated on the first execution of the job).
-storage_area <i>storage_name</i>	string	-	Identifies a storage area on which to run the report. Use the storage area's name. If this argument is not set, the report runs against all storage areas in the repository.
-output_device	string	-	Identifies a file to which to write the report data. The specification must be in the format: directory_path/file_name If the file already exists, data is appended to it. Use this option when you want to write directly to a tape drive or other device. If not set, the report file is saved to: System/Sysadmin/Reports/FileReport

Argument	Datatype	Default	Description
-report_renditions	Boolean	TRUE	When set to TRUE, rendition files are reported as well as the primary format files. Set to False if you do not wish to report renditions.
-sort_results	Boolean	TRUE	When set to TRUE, the file report is sorted by folder_path/object_name. Because this option requires a database sort of the entire data set returned, you may need to tune your database's sort/temp space parameters if you use this option.
-window_interval	integer	120	Execution window for the tool. Refer to The window interval, page 462 for a complete description.
-queueperson	string	-	Identifies the user who receives Inbox and email notifications from the tool.

Report sample

Each line of a File Report contains the following information:

- Document object_id
- Document's folder_path and object_name
- Document owner
- Document modification date
- Document version
- Content format
- Content page number
- Content file name

The following sample report describes a two-page Word document.

```
100015b4800001b5 /roger/research/newproject_1 roger
4/26/95 19:07:22 1.3 msw6 0
```

```

/u120/install/dmadmin/data/rogbase2/content_storage_01
/000015b4/80/00/01/49.doc
100015b4800001b5 /roger/research/newproject_1 roger
4/26/95 19:07:22 1.3 msw6 1

/u120/install/dmadmin/data/rogbase2/content_storage_01
/000015b4/80/00/01/4a.doc

```

The following sample report describes a single-page Word document with a PDF rendition.

```

090015b4800004f1 /roger/research/newproject_2
roger 6/16/95 20:00:47 1.7 msw6 0

/u120/install/dmadmin/data/rogbase2/content_storage_01
/000015b4/80/00/02/52.txt
090015b4800004f1 /roger/research/newproject_2 roger
6/16/95 20:00:47 1.7 pdf 0

/u120/install/dmadmin/data/rogbase2/
content_storage_01/000015b4/80/00/02/6e.pdf

```

Group Rename

The Group Rename tool re-names repository groups. This tool works in conjunction with Documentum Administrator. To rename a group, you must use the Groups screens in Documentum Administrator to identify the group and its new name. Documentum Administrator offers you two options for actually executing the rename operation:

- Running the Group Rename tool immediately after you identify the new name
- Queuing the operation until the next scheduled execution of the Group Rename tool

You cannot use the Set method to change a group name. You must go through Documentum Administrator and either a manual or automatic Group Rename execution to change a group name.

The Group Rename tool generates a report that lists the changes made to the repository objects for the group rename. The report is saved in the repository in `/System/Sysadmin/Reports/GroupRename`.

The tool is installed in the inactive state.

Arguments

The Group Rename tool has no arguments.

Index Agent Startup

The dm_FTIndexAgentBoot job starts index agents associated with a Content Server when that Content Server starts up. You do not need to run the job manually. Do not modify the dm_FTIndexAgentBoot job. Modifying the job is not supported.

Log Purge

The Log Purge tool deletes old log files. [Table 12-14, page 510](#), lists the log files deleted by Log Purge.

Table 12-14. Files deleted by the Log Purge tool

Log file or report	Location
Server log files	Documentum Server installation log location
dmbasic method server	Documentum Server installation log location
Connection broker log files	Documentum Server installation log location
Agent Exec log files	Documentum Server installation log location
Session log files	Documentum Server installation log location
Result log files	Temp cabinet
Job log files	Temp cabinet
Job reports	/System/Sysadmim/Reports folder
Lifecycle log files	Documentum Server installation log location
Method server log files	Documentum Server installation log location, MethodServer subdirectory

Result log files are generated by the execution of methods when the method's SAVE_RESULTS argument is set. Result log files are stored in Temp/Result.*method_name*.

Job log files are generated when a job is run. (The job log file for tools contains the job's trace file and the text of its report.) Job log files are stored in Temp/Jobs/*job_name*/log_*file*.

The lifecycle log files are generated when a lifecycle operation such as promote or demote occurs. The files are named bp_transition_*.log or bp_schedule_*.log, depending on the operation. They are stored in %\DOCUMENTUM%\dba\log*repository_id*\bp (\$DOCUMENTUM/dba/log/*repository_id*/bp).

Files are considered old and are deleted if they were modified prior to a user-defined cutoff date. By default, the cutoff date is 30 days prior to the current date. For instance, if you run Log Purge on July 27, all log files that were modified before June 28 are deleted.

You can change the cutoff interval by setting the `-cutoff_days` argument for the tool. (Refer to [Arguments](#), page 511, for instructions.)

Log Purge generates a report that lists all directories searched and the files that were deleted. The report is saved in the repository in `/System/Sysadmin/Reports/LogPurge`.

Note: If the tool is run at a remote distributed site, the report name has the site's server config name appended. For example, if London is a remote site, its report is found in `/System/Sysadmin/Reports/LogPurgeLondon`.

The Log Purge tool is installed in the inactive state.

Arguments

The Log Purge tool has three arguments, described in [Table 12-15](#), page 511.

Table 12-15. Log Purge arguments

Attribute	Datatype	Default	Description
<code>-cutoff_days</code>	integer	30	Controls what logs are deleted. All logs older than the specified number of days are deleted.
<code>-queueperson</code>	string(32)	-	User who receives email and inbox notifications from the tool. The default is the user specified in the <code>operator_name</code> attribute of the server config object.
<code>-window_interval</code>	integer	120	Execution window for the tool. Refer to The window interval , page 462 for a complete description.

Guidelines

Your business rules will determine how long you keep old log files and result log files. However, we recommend that you keep them at least 1 month as you may need them to debug a problem or to monitor the result of a method or job.

We recommend that you run this tool daily. This will ensure that your repository never has log files older than the number of days specified in the `cutoff_days` argument.

Report sample

The following is a sample of a Log Purge report. Its `start_date` attribute is set to June 3, 1996. The `cutoff_days` argument is set to 30 so that all logs older than 30 days will be deleted. The report looks for server and connection broker logs, session logs, and result logs from method objects and job objects (older than 30 days) and destroys them.

```
LogPurge Report For DocBase boston2
As Of 7/25/96 7:18:09 PM
Parameters for removing Logs:
-----
- Inbox messages will be queued to boston2
- Logs older than 30 days will be removed...

Looking for server and connection broker logs in the log
location...
Log Location: log
Log Location File Path: /u106/dm/dmadmin/dba/log
Changing directory to server log location:
/u106/dm/dmadmin/dba/log
Looking for session logs...
The top-level session log directory is:
/u106/dm/dmadmin/dba/log/00000962
Changing directory to:
/u106/dm/dmadmin/dba/log/00000962/boston2
Removing /u106/dm/dmadmin/dba/log/00000962/
boston2/01000962800008be
Removing /u106/dm/dmadmin/dba/log/00000962/
boston2/01000962800008e0
Removing /u106/dm/dmadmin/dba/log/00000962/
boston2/0100096280000904
Removing /u106/dm/dmadmin/dba/log/00000962/
boston2/0100096280000e28
Removing /u106/dm/dmadmin/dba/log/00000962/
boston2/0100096280000e72
Removing /u106/dm/dmadmin/dba/log/00000962/
boston2/0100096280000ed7
Changing directory to:
/u106/dm/dmadmin/dba/log/00000962/dmadmin
Removing /u106/dm/dmadmin/dba/log/00000962/
dmadmin/01000962800008b9
Removing /u106/dm/dmadmin/dba/log/00000962/
dmadmin/01000962800008ba
Removing /u106/dm/dmadmin/dba/log/00000962/
dmadmin/01000962800008b7
Removing /u106/dm/dmadmin/dba/log/00000962/
dmadmin/01000962800008b8
Changing directory to:
/u106/dm/dmadmin/dba/log/00000962/tuser3
Removing /u106/dm/dmadmin/dba/log/00000962/tuser3/
```

```
01000962800008fd
Changing directory to:
/u106/dm/dmadmin/dba/log/00000962/tuser1
Changing directory to:
/u106/dm/dmadmin/dba/log/00000962/agentexec
Removing /u106/dm/dmadmin/dba/log/00000962/agentexec/
agentexec.log.save.06.11.96.09.43.37
Changing directory to:
/u106/dm/dmadmin/dba/log/00000962/tuser4
Removing /u106/dm/dmadmin/dba/log/00000962/tuser4/
01000962800008fe
Removing /u106/dm/dmadmin/dba/log/00000962/tuser4/
0100096280000900
Removing /u106/dm/dmadmin/dba/log/00000962/tuser4/
0100096280000902
Removing /u106/dm/dmadmin/dba/log/00000962/tuser4/
0100096280000944
Removing /u106/dm/dmadmin/dba/log/00000962/tuser4/
0100096280000947
Removing /u106/dm/dmadmin/dba/log/00000962/tuser4/
0100096280000948
Removing /u106/dm/dmadmin/dba/log/00000962/tuser4/
0100096280000949
Removing /u106/dm/dmadmin/dba/log/00000962/tuser4/
010009628000094a
Removing /u106/dm/dmadmin/dba/log/00000962/tuser4/
010009628000094d
Removing /u106/dm/dmadmin/dba/log/00000962/tuser4/
010009628000094e
Removing /u106/dm/dmadmin/dba/log/00000962/tuser4/
0100096280000955
Removing /u106/dm/dmadmin/dba/log/00000962/tuser4/
trace.tmp
Changing directory to:
/u106/dm/dmadmin/dba/log/00000962/tuser2
Removing /u106/dm/dmadmin/dba/log/00000962/tuser2/
0100096280000e73
Changing directory to:
/u106/dm/dmadmin/dba/log/00000962/sysadmin
Changing directory to:
/u106/dm/dmadmin/dba/log/00000962/tuser5
Looking for result logs from dm_method objects...
Destroying Result.users_logged_in object
Destroying Result.users_logged_in object
Destroying Result.dmclean object
Destroying Result.dmclean object
Destroying Result.dmclean object
Destroying Result.dmclean object
Destroying Result.users_logged_in object
Destroying Result.users_logged_in object
Looking for result logs from dm_job objects...
Destroying 06/01/96 11:40:27 boston1 object
Destroying 05/31/96 11:40:41 boston1 object
Destroying 06/02/96 11:40:30 boston1 object
Destroying 06/03/96 11:40:15 boston1 object
Destroying 06/04/96 11:40:03 boston1 object
Destroying 06/19/96 11:40:00 boston1 object
```

```
Destroying 06/20/96 11:40:55 boston1 object
Destroying 06/22/96 11:40:32 boston1 object
Destroying 06/21/96 11:40:34 boston1 object
Destroying 06/24/96 11:40:10 boston1 object
Destroying 06/23/96 11:40:24 boston1 object
Destroying 06/25/96 11:40:07 boston1 object
Destroying 06/13/96 11:40:08 boston1 object
Destroying 06/16/96 11:40:18 boston1 object
Destroying 06/14/96 11:40:28 boston1 object
Destroying 06/15/96 11:40:27 boston1 object
Destroying 06/17/96 11:40:12 boston1 object
Destroying 06/18/96 11:40:07 boston1 object
Destroying 06/12/96 11:40:06 boston1 object
Destroying 06/11/96 11:40:11 boston1 object
Destroying 06/09/96 11:40:46 boston1 object
Destroying 06/08/96 11:40:03 boston1 object
Destroying 06/10/96 11:40:29 boston1 object
Destroying 06/06/96 11:40:01 boston1 object
Destroying 06/07/96 11:40:55 boston1 object
Destroying 06/05/96 11:40:02 boston1 object
Destroying 05/29/96 11:40:06 boston1 object
Destroying 05/30/96 11:40:49 boston1 object
```

```
End of Log Purge Report
Report End 7/25/96 7:19:13 PM
```

Queue Management

The Queue Management Tool deletes dequeued Inbox items. Whenever an item is queued to a user's Inbox, an object of type `dmi_queue_item` is created for that queued item. When users forward or otherwise remove an item from their Inboxes, the corresponding `dmi_queue_item` object is marked dequeued, but it is not removed from the repository. If these dequeued items are not removed, the tables for the `dmi_queue_item` type grow quite large, and performance degrades when users access their Inboxes. The Queue Management tool automates the task of removing these unneeded `dmi_queue_item` objects.

Which `dmi_queue_items` are removed is determined by the `cutoff_days` and `custom_predicate` arguments. The `cutoff_days` argument specifies the age of the objects you want to delete. The `custom_predicate` argument is applied to those items meeting the age requirement, allowing you to delete all or only some of them. For example, the tool could delete all dequeued `dmi_queue_items` that are older than 30 days and were queued to a specific user.

The tool generates a status report that provides you with a list of the deleted `dmi_queue_items`. The report is saved in the repository in `/System/Sysadmin/Reports/QueueMgt`.

If there is an error in the tool's execution, an email and Inbox notification is sent to the user specified by the `-queueperson` argument.

The Queue Management tool is installed in the inactive state.

Arguments

The Queue Management tool has four arguments, described in [Table 12-16, page 515](#).

Table 12-16. Queue Management arguments

Argument	Datatype	Default	Description
<code>-custom_predicate</code> <i>qualification</i>	string	-	<p>Defines a WHERE clause qualification for the query that selects dequeued items for deletion.</p> <p>The qualification must be a valid qualification and must work against the <code>dmi_queue_item</code> object. For example, a valid qualification is <code>"event='APPROVED'"</code> or <code>"name='dadmin'"</code>.</p> <p>Refer to The WHERE clause, page 142 in the <i>Content Server DQL Reference Manual</i> for complete information about WHERE clause qualifications.</p>
<code>-cutoff_days</code>	integer	90	<p>Defines a minimum age, in days, for dequeued items. All dequeued <code>dmi_queue_items</code> older than the specified number of days and that meet the specified qualification are deleted.</p> <p>To include all dequeued items in the search, set this value to zero (0).</p>

Argument	Datatype	Default	Description
-queueperson	string(32)	-	User who receives email and inbox notifications from the tool. The default is the user specified in the operator_name attribute of the server config object.
-window_interval	integer	120	Execution window for the tool. Refer to The window interval, page 462 for a complete description.

Note: The tool creates a base qualification that contains two conditions:

- delete_flag = TRUE
- dequeued_date = *value* (computed using cutoff_days argument)

Any qualification you add is appended to the base qualification.

Guidelines

Dequeued items are the result of moving objects out of an inbox. Objects are placed in inboxes by workflows, event notifications, archive and restore requests, or explicit Queue methods. Objects are moved out of an inbox when they are completed or delegated.

You must decide if there are any reasons to keep or maintain dequeued items. For example, you may want to keep dequeued items for auditing purposes. If so, leave this tool inactive or set the cutoff_days argument to a value that will save the dequeued items for a specified length of time.

After you have made your decisions, formulate a scheduling plan.

Note: The first time you execute the Queue Management tool, it may take a long time to complete if dequeued items have never been deleted before.

Report sample

Here is a sample of the report generated by the Queue Management tool.

```
QueueMgt Report For DocBase boston2
As Of 7/26/96 5:09:00 PM
Parameters for removing dequeued items:
-----
- Inbox messages will be queued to dmadmin
```

```
- No items dequeued before 7 days will be removed...
- The custom predicate is:
  name='tuser5'

Looking for dequeued items to delete...
Destroying queue item with ID 1b00096280000603
Destroying queue item with ID 1b000962800002e4
Destroying queue item with ID 1b00096280000304
Destroying queue item with ID 1b00096280000324
Destroying queue item with ID 1b00096280000344
5 dequeued items deleted...

End of Queue Management Report
Report End 7/26/96 5:09:00 PM
```

Remove Expired Retention Objects

The Remove Expired Retention Objects (RemoveExpiredRetnObjects) tool removes objects from the repository whose content, stored in a content-addressed storage area, has an expired retention date. The tool does not remove the actual content files or the associated content objects.

The tool invokes the CHECK_RETENTION_EXPIRED administration method to determine which SysObjects to remove from the repository. By default, the tool operates only on objects stored in content-addressed storage areas that require a retention date. You can also direct the tool to operate on content-addressed storage areas that allow but do not require a retention date by setting the INCLUDE_ZERO_RETENTION_OBJECTS argument. The tool never includes objects stored in content-addressed storage areas that do not allow retention periods. (Refer to the Guidelines for more information.)

After the tool runs the method to find the objects, it uses the Destroy method to remove them from the repository.

The tool generates a status report that provides you with a list of the deleted objects. The report is saved in the repository in /System/Sysadmin/Reports/RemoveExpiredRetnObjects. For each deleted object, the report lists the following attributes:

- r_object_id
- object_name
- a_storage_type
- r_creation_date
- retention_date

The retention_date attribute is a computed attribute.

The tool is installed in the inactive state.

Arguments

Table 12-17, page 518, describes the arguments for the tool.

Table 12-17. Remove Expired Retention Objects arguments

Argument	Datatype	Default	Description
<code>-query <i>qualification</i></code>	string	-	Identifies which objects are selected for possible removal. This is a DQL where clause qualification.
<code>-include_zero_retention_objects</code>	Boolean	F (FALSE)	Setting this to T (TRUE) directs the job to consider objects stored in a content-addressed storage area that allows but does not require a retention period.
<code>-queueperson</code>	string(32)	-	User who receives email and inbox notifications from the tool. The default is the user specified in the <code>operator_name</code> attribute of the server config object.
<code>-window_interval</code>	integer	1440	Execution window for the tool. Refer to The window interval, page 462 for a complete description.

Guidelines

A content-addressed storage area can have three possible retention period configurations:

- The storage area may require a retention period.
In this case, the `a_retention_attr` name attribute is set and the `a_retention_attr_req` is set to T.
- The storage area may not allow a retention period.
In this case, the `a_retention_attr` name attribute is not set and the `a_retention_attr_req` is set to F.
- The storage area may allow but not require a retention period.

In this case, the `a_retention_attr` name attribute is set, but the `a_retention_attr_req` is set to F.

By default, the method does not include objects whose content has a 0 retention period because the assumption is that such content is meant to be kept forever. However, in a storage area that allows but does not require a retention period, a 0 retention period can be result from two possible causes:

- The user deliberately set no retention period, and consequently, the server set the retention period to 0
- The user specified a retention date that had already elapsed. When this occurs, the server sets the retention period to 0.

Because the meaning of 0 is ambiguous in such storage areas, the tool supports the `INCLUDE_ZERO_RETENTION_OBJECTS` argument to allow you to include content with a zero retention in storage areas that allow but do not require a retention period.

If you set `INCLUDE_ZERO_RETENTION_OBJECTS` to T, when the tool examines objects in storage areas that allow but do not require a retention period and it will remove from the repository any object with an expired or zero retention period. the tool does not remove the actual content files or associated content objects. (You must run `dmclean` to remove those.)

Refer to [CHECK_RETENTION_EXPIRED](#), page 182, in the *Content Server DQL Reference Manual* for further information about the underlying method.

Report sample

```

--- Start C:\Documentum\dba\log\00002710\sysadmin\
RemoveExpiredRetnObjectsDoc.txt report output ----
RemoveExpiredRetnObjects Report For DocBase dctm52
  As Of 2/26/2004 15:39:10

RemoveExpiredRetnObjects utility syntax:
apply,c,NULL,CHECK_RETENTION_EXPIRED,
QUERY,S,'a_storage_type = ''destroy_test3''
,INCLUDE_ZERO_RETENTION_OBJECTS,B,T
Executing RemoveExpiredRetnObjects...
Object 090027108000c910 destroyed successfully.
Object 090027108000c911 destroyed successfully.
# of objects with expired retention that matched
the condition: 2
# of successfully destroyed: 2
# of objects not destroyed : 0
Report End 2/26/2004 15:39:14
--- End C:\Documentum\dba\log\00002710\sysadmin\
RemoveExpiredRetnObjectsDoc.txt report output ---

```

Rendition Manager

The Rendition Manager tool removes unwanted renditions of versioned documents. A rendition is a copy of a document's content in a different format than the original. Renditions, like the original content files, are stored in storage areas. Over time, unneeded renditions from previous versions of documents can take up noticeable amounts of disk space. (For information about renditions, refer to [Creating renditions, page 184](#), of *Content Server Fundamentals*.)

The tool's arguments define which renditions are removed. The tool can delete renditions based on their age, format, or source (client- or server-generated). The tool removes the content objects associated with unwanted renditions. The next execution of the Dmclean tool automatically removes the renditions' orphaned content files (assuming that Dmclean's clean_content argument is set to TRUE).

Note: Renditions with the page modifier dm_sig_template are never removed by Rendition Manager. These renditions are electronic signature page templates; they support the electronic signature feature available with Trusted Content Services.

The report generated by the tool lists the renditions targeted for removal. The report is saved in the repository in /System/Sysadmin/Reports/RenditionMgt.

The Rendition Manager tool is installed in the inactive state.

Arguments

[Table 12-18, page 520](#), describes the arguments for the Rendition Management tool.

Table 12-18. Rendition Management arguments

Argument	Datatype	Default	Description
-keep_labels	Boolean	TRUE	Indicates whether you wish to keep renditions with symbolic labels. When this is TRUE, the -keep_current argument is ignored because CURRENT is a symbolic label.

Argument	Datatype	Default	Description
-keep_current	Boolean	TRUE	Indicates whether you wish to keep renditions of documents with the symbolic label CURRENT. When this is TRUE, the CURRENT version is always kept even if -keep_slable is set to FALSE.
-keep_keep_flag	Boolean	TRUE	<p>Controls whether content objects with a rendition attribute value of 3 are deleted by the tool. T (TRUE) instructs the tool to not delete the objects.</p> <p>The default value is F (FALSE), meaning the content objects are not deleted.</p> <p>Note: The rendition attribute in a content object is set to 3 when a rendition is added to content with the keep_rendition argument in the Addrendition method set to T (TRUE).</p>
-keep_esignature	Boolean	TRUE	<p>Controls whether renditions with the page modifier dm_sig_source are removed.</p> <p>T (TRUE) means to keep those renditions. F (FALSE) means to remove those renditions.</p>
-cutoff_days	integer	180	The maximum age, in days, of renditions that you want to keep. All renditions older than the specified number of days are considered for deletion.

Argument	Datatype	Default	Description
-server_renditions	string	all	<p>Specifies which server-based renditions to remove. Valid values are:</p> <p>all none</p> <p>or a list of formats</p> <p>If a list of formats is specified, the format names must be separated by single spaces.</p> <p>The default is all.</p>
-client_renditions	string	none	<p>Specifies which client renditions to remove (includes the renditions added using the Addrendition method). Valid values are:</p> <p>all none</p> <p>or a list of formats</p> <p>If a list of formats is specified, the format names must be separated by single spaces.</p> <p>The default is none.</p>
-report_only	Boolean	TRUE	<p>Indicates whether to generate only a report and not delete renditions. Set this to FALSE if you want to actually delete the renditions in addition to generating a report.</p>

Argument	Datatype	Default	Description
-queueperson	string(32)	-	User who receives email and inbox notifications from the tool. The default is the user specified in the operator_name attribute of the server config object.
-window_interval	integer	120	Execution window for the tool. Refer to The window interval , page 462 for a complete description.

Guidelines

The Rendition Management tool removes all renditions that meet the specified conditions and are older than the specified number of days. For example, if you execute this tool on July 30 and the -cutoff_days argument is set to 30, then all renditions created or modified prior to June 30 are candidates for deletion. On July 31, all renditions created or modified before July 1 are removed.

We recommend that you run this tool with the -report_only argument set to TRUE first to determine how much disk space renditions are using and what type of renditions are in the repository. With -report_only set to TRUE, you can run the tool several times, changing the other arguments each time to see how they affect the results.

We recommend that you have a thorough knowledge of how and when renditions are generated before removing any. For example, client renditions, such as those added explicitly by an Addrendition method, may be difficult to reproduce if they are deleted and then needed later. Content Server renditions are generated on demand by the server and are generally easier to reproduce if needed.

To ensure that your repository never has rendition files older than the number of days specified in the -cutoff_days argument, run this tool daily.

Note: The first time you execute the Rendition Management tool, it may take a long time to complete if the old renditions have never been deleted before.

Report sample

Here is a sample of the Rendition Management's report.

```
RenditionMgt Report For DocBase boston2
```

```

As Of 7/27/96 3:57:01 PM
Parameters for removing renditions:
-----
- Inbox messages will be queued to dmadmin
- No renditions accessed in the last 0 days will
  be removed...
- Renditions of documents having symbolic labels
  will be removed...
- Renditions for the current version will NOT
  be removed...
- This will generate a report only; no renditions
  will be removed...
- The following client renditions are candidates
  for removal:
    1) mactext
- The following server renditions are candidates
  for removal:
    1) crtext

NOTE: This is a report only - no renditions will be
      removed

Querying for renditions...
Object NameOwner Name FormatAccess DateRendition Type
RenditionMgtdmadmincrtext07/27/96 15:44:31server
RenditionMgtdmadmincrtext07/27/96 15:45:31server
RenditionMgtdmadmincrtext07/27/96 15:45:31server
teststatus2boston2crtext07/24/96 18:40:23server
SwapInfodmadmincrtext07/11/96 16:16:16server
DBWarningdmadmincrtext07/12/96 18:31:11server
foo717bbbdmadmincrtext07/24/96 20:48:46server
ContentWarningdmadmincrtext07/18/96 18:01:19server
...
...
SwapInfodmadmincrtext07/27/96 13:20:47server
RenditionMgtdmadmincrtext07/27/96 15:48:09server
rend722 dmadminmactext07/02/96 11:00:11client/external
rend722dmadminmactext07/02/96 11:02:13client/external
rendzz2dmadminmactext07/02/96 11:05:04client/external
rendyy2dmadminmactext07/02/96 12:16:10client/external
rendww2dmadminmactext07/02/96 12:27:05client/external
foor717atuser4mactext07/17/96 17:48:25client/external
foor725atuser5mactext07/25/96 12:05:07client/external
Report Summary:
-----
    The Docbase has a total of 39,216 kbytes of content.
    54 renditions were reported.
    The renditions reported represent 82 kbytes of content
    or 0.21%
End of Rendition Management Report
Report End 7/27/96 3:57:10 PM

```

State of the Repository Report

The State of the Repository Report tool generates a report to help you troubleshoot repository problems. A partial list of the information included in the report is:

- The attribute values in the docbase config object
- Server initialization information from the server.ini file
- The directory paths defined by the location objects in the server config object
- Version numbers of your server, RDBMS, and operating system

The report is saved in the repository in /System/Sysadmin/Reports/StateOfRepository.

The State of the Repository tool is installed in the active state.

Arguments

The State of the Repository tool has one argument, described in [Table 12-19, page 525](#).

Table 12-19. State of the Repository arguments

Attribute	Datatype	Default	Description
-window_interval	integer	720	Execution window for the tool. Refer to The window interval, page 462 for a complete description.

Report sample

Here is a sample report from the State of the Repositories tool. The example shows all of the sections in the tool's report, but truncates entries in some sections in the interests of space.

```
StateOfRepository Report For Repository dm_master As Of 4/12/1999 09:26:32

Repository Configuration:
Description:The Test Repository
Federation Name:<dm_master is not in a Federation>
Repository ID:22000
Security Modeacl
Folder Security:On
Authorization Protocol:<Not defined>
Database:SQLServer
RDBMS Index Store:<Not defined>
Mac Access Protocol:none
```

```

Server Configuration:
Server Name:dm_master
Server Version:4.0 Win32.SQLServer7
Default ACL:Default ACL of User
Host Name:bottael
Install Owner:dadmin
Install Domain:bottael
Operator Name:dm_master
Agent Launcher:agent_exec_method
Checkpoint Interval:300 seconds
Compound Integrity:On - Server enforces integrity for virtual documents
Turbo Backing Store:filestore_01
Rendition Backing Store:<Not defined>
Web Server Location:BOTTAE1
Web Server Port:80
Rightsite Image:/rs-bin/RightSit

Server Locations:
events_locationD:\DOCUMENTUM\share\data\events
common_locationD:\DOCUMENTUM\share\data\common
temp_locationD:\DOCUMENTUM\share\temp
log_locationD:\DOCUMENTUM\dba\log
system_converter_location D:\DOCUMENTUM\product\4.0\convert
user_converter_location<Not defined>
verity_locationD:\DOCUMENTUM\product\4.0\verity
user_validation_location <Not defined>
assume_user_locationD:\DOCUMENTUM\dba\dm_assume_user.exe
change_password_locationD:\DOCUMENTUM\dba\dm_change_password.exe
signature_chk_locD:\DOCUMENTUM\dba\dm_check_password.exe
stage_destroyer_location<Not defined>

Set Information:

CLASSPATH=C:\PROGRA~1\DOCUME~1\DFCRE40\lib\dfc.jar;C:\PROGRA~1\DOCUME~1\Classes;
C:\Netscape\ldapjava\packages
COLORS=white on blue
COMPUTERNAME=BOTTAE1
ComSpec=C:\WINNT\system32\cmd.exe
CVSROOT=godzilla.documentum.com:/docu/src/master
CVS_SERVER=cvs1-9
DM_HOME=D:\DOCUMENTUM\product\4.0
DOCUMENTUM=D:\DOCUMENTUM
HOMEDRIVE=C:
HOMEPATH=\
LOGONSERVER=\\BOTTAE1
NUMBER_OF_PROCESSORS=1
OS=Windows_NT
Os2LibPath=C:\WINNT\system32\os2\dll;
Path=D:\DOCUMENTUM\product\4.0\bin;C:\PROGRA~1\DOCUME~1\DFCRE40\bin;
D:\DOCUMENTUM\product\3.1\bin;C:\WINNT\system32;C:\WINNT;
c:\program files\maestro.nt;C:\PROGRA~1\DOCUME~1\Shared;
c:\SDK-Java.201\Bin;c:\SDK-Java.201\Bin\PackSign;c:\Winnt\Piper\dll;
c:\Program Files\DevStudio\SharedIDE\bin;
c:\Program Files\DevStudio\SharedIDE\bin\ide;D:\MSSQL7\BINN;
c:\cvs1.9;c:\csh\bin;c:\csh\samples;C:\DOCUME~1\RIGHTS~1\product\bin
PATHEXT=.COM;.EXE;.BAT;.CMD
PROCESSOR_ARCHITECTURE=x86
PROCESSOR_IDENTIFIER=x86 Family 5 Model 2 Stepping 5, GenuineIntel
PROCESSOR_LEVEL=5

```

```

PROCESSOR_REVISION=0205
PROMPT=$P$G
SystemDrive=C:
SystemRoot=C:\WINNT
TEMP=C:\TEMP
TMP=C:\TEMP
USERDOMAIN=BOTTAE1
USERNAME=dmadmin
USERPROFILE=C:\WINNT\Profiles\dmadmin
windir=C:\WINNT

```

Registered tables in the Repository:

Table NameTable OwnerOwner:Group:World Permits

```

adm_turbo_sizedbo 1:1:1
dm_federation_logdbo 15:7:3
dm_portinfodbo 1:1:1
dm_queuedbo 1:1:1

```

Number of Documents by Type:

Document Type	Count
dmi_expr_code	74
dm_method	66
dm_document	44
dm_folder	31
dm_job	29
dm_location	14
dm_registered	14
dm_procedure	10
dm_cabinet	6
dm_script	3
dm_smart_list	2
dm_business_pro	1
dm_docbase_config	1
dm_mount_point	1
dm_outputdevice	1
dm_query	1
dm_server_config	1
Total:	-----
	299

Number of Documents by Format:

Document Format	Count
crtext	11
mdoc55	9
maker55	5
<NO CONTENT>	2
msw8template	2
vrf	2
wp6	2
excel5book	1
excel8book	1
excel8template	1

```

ms_access7                1
ms_access8_mde            1
msw6                      1
msw8                      1
powerpoint                1
ppt8                     1
ppt8_template             1
ustn                      1
Total:                    -----
                              44

```

Number of Documents by Storage Area:

```

Storage Area              Count

filestore_01              40
dm_turbo_store            4
Total:                    -----
                              44

```

Content Size(KB) by Format:

FormatLargestAverageTotal

```

mdoc5515885772
crtext 10115436
text19621254
vrf 192119238
maker553722114

```

Content Size(KB) by Renditions:

FormatLargestAverageTotal

Content Size(KB) Summary:

```

filestore_01
  Largest Content:          196
  Average Content:         31
  Total Content:           2,092

dm_turbo_store
  Largest Content:          8
  Average Content:         5
  Total Content:           34
-----
GTtotal Content:           2,127
GTtotal Rendition:        (0.00% of total content)

```

Number of Users and Groups:

```

Named Users                4
Groups                     2

```

ACL Summary:

```

Number of ACLs:            33
Number of Internal ACLs:   29
Number of External System ACLs: 4
Number of External Private ACLs:

```

Report End 4/12/1999 09:26:54

Swap Info

Note: The Swap Info tool is not installed if Content Server is running on an HPUX machine.

The Swap Info tool uses operating system commands to retrieve information about swap space usage and availability. The tool generates a report but does not issue warnings because there is no realistic way to determine if the swap space is too low as this determination has too many variables. The status report is saved in the repository in `/System/Sysadmin/Reports/SwapInfo`.

Note: If the tool was run at a remote distributed site, the report name will have the site's server config name appended. For example, if London is a remote site, its report would be found in `/System/Sysadmin/Reports/SwapInfoLondon`.

The Swap Info tool is installed in the active state.

Arguments

The Swap Info tool has two arguments, described in [Table 12-20, page 529](#).

Table 12-20. Swap Info arguments

Argument	Datatype	Default	Description
<code>-queueperson</code>	string(32)	-	User who receives email and inbox notifications from the tool. The default is the user specified in the <code>operator_name</code> attribute of the server config object.
<code>-window_interval</code>	integer	120	Execution window for the tool. Refer to The window interval, page 462 for a complete description.

Report sample

The format of the report generated by Swap Space Info varies by operating system. Here is a sample of a report run against the Solaris operating system:

```
SwapInfo Report For DocBase boston2
As Of 7/26/96 4:00:59 PM
Summary of Total Swap Space Usage and Availability:

total: 59396k bytes allocated + 49216k reserved =
      108612k used, 287696k available
Swap Status of All Swap Areas:
swapfile          dev swaplo blocks  free
/dev/dsk/c0t3d0s1 32,25      8 717688 626640
Report End 7/26/96 4:00:59 PM
```

ToolSetup

ToolSetup is an installation utility for the system administration tools. In a non-distributed environment, the tools are installed as part of the installation procedure.

You can rerun ToolSetup as often as needed, to install new versions of the software or until all tools are installed at all sites in a multiple site configuration. Running ToolSetup does not affect previously installed system administration tools or their argument values.

You must run ToolSetup from the primary site (where the RDBMS resides). To install tools in component sites, use the setupdist utility.

You can run Toolsetup from the command line or using Documentum Administrator.

To run ToolSetup from the command line:

1. At the primary site, log in to the %DOCUMENTUM%\install\admin (\$DOCUMENTUM/install/admin) directory.
2. Execute the following command for the primary site:

```
dmbasic -ftoolset.ebs -eToolSetup - -
<><current_path>
```

Update Statistics

The Update Statistics tool generates current statistics for the RDBMS tables owned by the repository owner. Generating statistics is always useful, particularly after you perform load operations or if table key values in the underlying RDBMS tables are not normally distributed.

When you run the tool against an Oracle or Sybase database, the tool uses a file that contains commands to tweak the database query optimizer. For Oracle, the file is named `custom_oracle_stat.sql`. For Sybase, it is named `custom_sybase_stat.sql`. The file is stored in `%DOCUMENTNUM %\dba\config\repository_name` (`$DOCUMENTNUM /dba/config/repository_name`). You can add commands to this file. However, do so with care. Adding to this file affects query performance. If you do add a command, you can use multiple lines, but each command must end with a semi-colon (;). You cannot insert comments into this file.

The `-dbreindex` argument controls whether the method also reorganizes database tables in addition to computing statistics. For SQL Server and DB2, you can set the argument to either `READ` or `FIX`. Setting it to `READ` generates a report on fragmented tables but does not fix them. Setting it to `FIX` generates the report and fixes the tables. (In either case, the report is included in the overall job report.)

For Sybase, the `-dbreindex` argument is only effective if set to `FIX`, to reorganize the tables. Setting it to `READ` does not generate a report on Sybase. If you include the `-dbreindex` argument set to `FIX`, the repository owner (the account under which the tool runs) must have `sa` role privileges in the database.

The `-dbreindex` argument has no effect on a Oracle database.

The tool generates a report that is saved in the repository in `System/Sysadmin/Reports/UpdateStats`. The exact format of the report varies for each database.

The Update Statistics tool is installed in the active state, running once a week. Because this tool can be CPU and disk-intensive, it is recommended that you run the tool during off hours for database use. Consult with your RDBMS DBA to determine an optimal schedule for this tool.

Arguments

The Update Statistics tool has three arguments, described in [Table 12-21, page 531](#).

Table 12-21. Update Statistics arguments

Argument	Datatype	Default	Description
<code>-dbreindex</code>	string	READ	Controls whether the tool actually updates statistics or only reports on RDBMS tables that need updating. READ generates only the report. This setting is valid only for SQL Server and DB2 databases.

Argument	Datatype	Default	Description
			<p>FIX generates the report and updates the tables. This setting is valid on SQL Server, DB2, and Sybase databases. However, on Sybase, it only fixes the tables. A report is not generated.</p> <p>This argument is not available for Oracle databases.</p>
-server_name	string(32)	-	<p>Name of the database server.</p> <p>This is a required argument on SQL Server and Sybase. It is set for the job when the administration tools are installed in repositories running against a SQL Server or Sybase database.</p>
-queueperson	string(32)	-	<p>User who receives email and inbox notifications from the tool. The default is the user specified in the operator_name attribute of the server config object.</p>
-window_interval	integer	120	<p>Execution window for the tool. Refer to The window interval, page 462 for a complete description.</p>

Guidelines

Run this tool after you perform large loading operations.

When the job is run with -dbreindex set to READ and the statistics need updating, the report will say:

```
-dbreindex READ. If rows appear below, the corresponding
tables are fragmented.
Change to -dbreindex FIX and rerun if you want to reindex
these tables.
```

When the job is run with `-dbreindex` set to `FIX`, the report will say:

```
-dbreindex FIX. If rows appear below, the corresponding
tables have been reindexed.
Change to -dbreindex READ if you don't want to reindex
in the future.
```

Report sample

The Update Statistics report tells you when the tool was run and which tables were updated. The report lists the update statistics commands that it runs in the order in which they are run. Here is a sample of the report:

```
Update Statistics Report:

Date of Execution: 06-04-96

update statistics dmi_object_type
go
update statistics dm_type_s
go
update statistics dm_type_r
go
update statistics dm_type_r
go
update statistics dmi_index_s
go
. . .
End of Update Statistics Report
```

User Chg Home Db

The `UserChgHomeDb` tool changes a user's home repository. This job works in conjunction with Documentum Administrator. To change a user's home repository, you must connect to the repository using Documentum Administrator and make the change through the Users screens. Documentum Administrator gives you two options for performing the change:

- Execute the `UserChgHomeDb` tool immediately after you save the change
- Queue the change, to be performed at the next scheduled execution of `UserChgHomeDb`.

You cannot change a user's home repository using the `Set` method. When a user's home repository changes, the change must be cascaded to several other objects that make use of it.

The UserChgHomeDb tool generates a report that lists the objects changed by the operation. The report is saved in the repository in /System/Sysadmin/Reports/UserChgHomeDb.

The User Chg Home Db tool is installed in the inactive state.

Arguments

The User Chg Home Db tool has no arguments.

User Rename

The User Rename tool changes a user's repository name. This job works in conjunction with Documentum Administrator. To change a user's name, you must connect to the repository using Documentum Administrator and make the change through the Users screens. Documentum Administrator gives you two options for performing the change:

- Execute the User Rename tool immediately after you save the change
- Queue the change, to be performed at the next scheduled execution of User Rename.

You cannot change a user's name using the Set method. When a user's name changes, the change must be cascaded to many other objects that make use of it.

The User Rename tool generates a report that lists the objects changed by the operation. The report is saved in the repository in /System/Sysadmin/Reports/UserRename.

The User Rename tool is installed in the inactive state.

Arguments

The User Rename tool has no arguments.

Report sample

This sample report was generated when the tool was run in Report Only mode.

```
Job: dm_UserRename
Report For Repository example.db_1;
As Of 3/6/2000 12:00:27 PM

=====3/6/2000 12:00:28 PM=====
Reporting potential changes in repository example.db_1
```

```

when renaming user 'dm_autorender_mac' to 'test'.

The following user rename options were specified:
Execution Mode: Report Only
Checked out Objects: Unlock
WARNING: There are 15 sessions currently open. It is
recommended that user rename is performed in single
user connection mode.

===== DM_USER OBJECT =====
Object type : dm_user
Object id : 1100162180000103
Name : dm_autorender_mac
Attributes referencing user dm_autorender_mac: user_name
-----
==== ACL Objects referencing user dm_autorender_mac ====
-----
Object type : dm_acl
Object id : 450016218000010c
Name : dm_450016218000010c
Attributes referencing user dm_autorender_mac:
  owner_name
-----
**** Number of ACL objects affected: 1

==== Alias Set Objects referencing user dm_autorender_mac
**** Number of Alias Set objects affected: 0

==== Dm_user object. Default ACL of user object is
referencing dm_autorender_mac
-----
Object type : dm_user
Object id : 1100162180000103
Name : dm_autorender_mac
Attributes referencing user dm_autorender_mac: acl_domain
-----
==== Sysobjects referencing user 'dm_autorender_mac',
which are not locked
**** Number of sysobjects affected: 0

==== Sysobject referencing user 'dm_autorender_mac',
which are locked (all the objects in this list will be
unlocked and modified)
**** Number of sysobjects affected: 0

==== Sysobjects locked by user 'dm_autorender_mac' ==
(all the objects in this list will be unlocked)
**** Number of sysobjects affected: 0

==== Routers referincing user dm_autorender_mac. ====
**** Number of router objects affected: 0

==== Workflow objects referincing user
dm_autorender_mac.
**** Number of dm_workflow objects affected: 0

==== Activity objects referincing user
dm_autorender_mac.

```

```
**** Number of dm_activity objects affected: 0

===== Workitem objects referincing user
dm_autorender_mac.
**** Number of dmi_workitem objects affected: 0

===== Groups referincing user dm_autorender_mac. ===
-----
Object type : dm_group
Object id : 1200162180000100
Name : docu
Attributes referencing user dm_autorender_mac:
users_names[2]
-----
**** Number of group objects affected: 1

===== dmi_queue_item objects referincing user
dm_autorender_mac.
**** Number of dmi_queue_item objects affected: 0

===== dmi_registry objects referincing user
dm_autorender_mac.
**** Number of dmi_registry objects affected: 0

===== The following dm_registered objects have table_owner
attribute referencing user dm_autorender_mac. The script
will not update the objects. You have to modify them manually.

===== The following dm_type objects have owner_name
attribute referencing user dm_autorender_mac. The scrip
t will not update the objects. You have to modify them
manually.

===== The following dmi_type_info objects have acl_domain
attribute referencing user dm_autorender_mac. The script
will not update the objects. You have to modify them manually.

-----3/6/2000 12:00:28 PM-----
```

Version Management

The Version Management tool removes unwanted versions of documents from the repository. This tool automates the Destroy and Prune methods. (Refer to [Versioning, page 118](#), of *Content Server Fundamentals* for a discussion of how Documentum's versioning works. Refer to [Destroy, page 186](#), and [Prune, page 326](#), in the *Content Server API Reference Manual* for descriptions of the Destroy and Prune methods.)

Note: The tool removes only the repository object. It does not remove content files associated with the object. To remove content files, use the Dmclean tool, described in [Dmclean, page 494](#).

The arguments you define for the tool determine which versions are deleted. For example, one argument (`keep_labels`) lets you choose whether to delete versions that have a symbolic label. Another argument (`custom_predicate`) lets you define a WHERE clause qualification to define which versions are deleted. (Refer to [The WHERE clause, page 142](#), in the *Content Server DQL Reference Manual* for instructions on writing WHERE clauses.)

The Version Management tool generates a status report that is saved in the repository in `/System/Sysadmin/Reports/VersionMgt`.

The Version Management tool is installed in the inactive state.

Arguments

The Version Management tool arguments are described in [Table 12-22, page 537](#).

Table 12-22. Version Management arguments

Argument	Datatype	Default	Description
<code>-keep_labels</code>	Boolean	TRUE	Indicates whether you wish to keep versions of documents with symbolic labels. The default is TRUE.
<code>-custom_predicate</code> <i>qualification</i>	string	-	<p>Defines a WHERE clause qualification for the query that selects versions for deletion.</p> <p>The qualification must be a valid qualification. Refer to The WHERE clause, page 142 of the <i>Content Server DQL Reference Manual</i> for information about WHERE clause qualifications.</p>

Argument	Datatype	Default	Description
-cutoff_days	integer	180	The maximum age, in days, of the versions that you want to keep. All versions older than the specified number of days are considered for deletion. If you set this flag, you cannot set the -keep_latest flag. The two flags are mutually exclusive.
-keep_latest	integer	-	Directs the tool to keep the specified number of versions directly derived from each end node of a version branch. If you set this flag, you cannot set the -cutoff_days flag. The two flags are mutually exclusive.
-report_only	Boolean	TRUE	Indicates whether to generate only the report. Set this to FALSE to actually remove versions.
-queueperson	string(32)	-	User who receives email and inbox notifications from the tool. The default is the user specified in the operator_name attribute of the server config object.
-window_interval	integer	120	Execution window for the tool. Refer to The window interval, page 462 for a complete description.

Guidelines

We recommend that you have a thorough knowledge of what prior versions mean for your business requirements. If you need to keep all versions to satisfy auditing requirements, do not run this tool at all. Individual users or departments may also have needs and requirements for older versions. Needs for disk space may also affect the decisions about how many older versions to keep.

Run this tool initially with the `-report_only` argument set to `TRUE` to determine how much disk space versions are using and how many versions are in the repository. With `-report_only` set to `TRUE`, you can run the report several times, changing the other arguments each time to see how they affect the results.

If you are using the `-cutoff days` argument to ensure that your repository never has only versions older than a specified number of days, run this tool daily. If you are using `-keep_latest` argument to keep only a specified number of versions, you can run this tool less frequently. The frequency will depend on how often new versions are generated (thereby necessitating the removal of old versions to keep the number of versions constant).

You can use this tool for one-time events also. For example, after a project is completed, you might remove all older versions of the project's documentation. (You must set the arguments appropriately for such occasions and reset them after the job is finished.)

Note: the first time you execute the Version Management tool, it may take a long time to complete if the old versions have never been deleted before.

Report sample

```
VersionMgt Report For DocBase boston2
As Of 9/12/96 11:33:33 AM
Parameters for deleting versions:-----
- Inbox messages will be queued to boston2
- Keep the 3 most recent versions of each version
tree...
- Documents having symbolic labels will NOT be
deleted...
- This will generate a report and delete the
versions...
- The custom predicate is:
  object_name='ver911c'
- The server is enforcing compound integrity
(the compound_integrity attribute in the Server
  Config object is set to true).

Querying for versions...
Object Name   Owner Name   Modify Date      Version Labels
ver911c       tuser1       09/12/96 11:19:30 1.7.1.2
ver911c       tuser1       09/12/96 11:18:49 1.7.1.0
ver911c       tuser1       09/11/96 16:18:46 1.3.1.1
ver911c       tuser1       09/11/96 16:18:37 1.3.1.0
ver911c       tuser1       09/11/96 16:07:40 1.5
ver911c       tuser1       09/11/96 16:07:39 1.4
ver911c       tuser1       09/11/96 16:07:37 1.1
ver911c       tuser1       09/11/96 16:07:35 1.0
Report Summary:
-----
The Docbase has a total of 21,986 kbytes of content.
8 versions were removed.
```

```
The versions removed represented 12 kbytes of content
or 0.05%
The documents contents are now orphaned. Use the
Dmclean system administration tool to actually remove
the contents.
```

Tool maintenance and troubleshooting

The tool suite typically requires very little maintenance after you define execution schedules and arguments for the tools.

Instructions for setting the scheduling attributes are found in [Activating and scheduling administration tools, page 465](#).

Changing the default settings

The jobs that make up the tool suite are installed with default attribute values that may not be appropriate for your environment. You may want to change the default settings of a job after you have:

- Examined report results
- Determined warning thresholds that are appropriate for your site
- Defined a staggered execution schedule for the tools
- Defined qualifying arguments for the destructive tools

The easiest way to change the attributes for a tool's job is using Documentum Administrator. However, you can also use DQL or the API to change a tool's settings.

The server always passes the standard arguments to the tools. The methods associated with the tools use the `job_id` argument to obtain the remaining tool-specific arguments from the job object. Do not reset this argument for tools.

Using DQL

To view a tool's attributes, use the `SELECT` statement to retrieve the attribute values in which you are interested.

To change a tool's attributes, use the `UPDATE...OBJECT` statement. To ensure that you update the correct job object, qualify your query with a condition that uniquely identifies the job you want to change.

For information about these statements, refer to [Select, page 115](#), and [Update...Object, page 158](#), in the *Content Server DQL Reference Manual*.

Using API

To use API methods to change a tool's attributes:

1. Connect to the repository containing the job.
2. Fetch the job that implements the tool.
3. Change the attributes that you want to modify.
4. Save the job object.

Using the tool trace log files

If an error occurs during the execution of a tool, the queueperson is notified by Inbox and email messages. Generally, the message indicates a failure in the method and directs you to the appropriate trace log file. The trace log files are stored in the Temp cabinet as `Result.method name`. For example, the log file for the Queue Management tool is called `Result.dm_QueueMgt`. Examine the log files to determine the cause of the failure. A trace log file is generated whenever a tool is executed.

Viewing the tool reports

You can view the tool reports using Documentum Administrator or by opening the repository folder in which they are contained. The reports provide historical accounts of your repository environment and should be examined regularly so that you can respond in a timely fashion to any problems or administrative needs that the tools uncover.

To view the reports directly in the repository, open the Sysadmin folder in the System cabinet. The reports are in a subfolder called Reports.

Tracing

Tracing instructs the server to log specific information in a log file. Use this information for troubleshooting.

You must have Sysadmin or Superuser privileges to turn tracing on and off.

Types of tracing

The areas for tracing are:

- Server based
- Session based
- Client based

Trace messages are sent to different types of log files:

- The server log file records root server activities.
- The session log file records all SQL commands generated from DQL commands and other messages determined by the specified severity level. Informational, warning, error, and fatal error messages are always written to the session log file, even if tracing is turned off.
- The api.log file, on the client, records all user API method calls and the responses generated by the API.

The agent exec process generates generate trace log files for all jobs. The log files are put in `Temp/Jobs/job_name/job_name_and_date`. The `job_name_and_date` is a concatenation of the name of the job that generated the report and the date on which the report was generated. Each time the job runs, a new file is generated. Previous files are not overwritten.

To delete these files, you can use DQL:

```
DELETE dm_document OBJECT
WHERE FOLDER('/Temp/Jobs/jobname')
```

Substitute the job's name for *jobname*.

For more information about the trace log files for jobs, refer to [Using the tool trace log files, page 541](#).

Server tracing

Content Server writes server tracing information to the server log file.

You can turn on server tracing in two ways:

- Using trace flags at server start-up
- Using the Apply method after server start-up

You can use the Apply method to turn off Server tracing.

Server start-up tracing options

Only the last SQL statement tracing is turned on by default when a server starts. To start tracing other options at server start-up, specify the trace flags for the options with the -o option in the command line. The name of the option must immediately follow the -o. No space is allowed between the -o and the option name. On Windows platforms, add the -o option to the command line by editing Content Server's service entry.

Table 12-23, page 543, lists the trace flags that you can use with the -o option at server start-up:

Table 12-23. Server start-up trace options

Option name	Captures this information in the server log
debug	Session shutdown, change check, launch and fork information
docbroker_trace	connection broker information
i18n_trace	Session locale and client code page usage
lock_trace	Windows locking information
net_ip_addr	IP addresses of client and server for authentication
nettrace	Turns on RPC tracing (traces Netwise calls, connection ID, client host address, and client host name)
sqltrace	SQL commands sent to the underlying RDBMS for subsequent sessions
ticket_trace	Traces import and export operations for login ticket keys and operations using single-use login tickets
trace_authentication	Detailed authentication information
trace_complete_launch	Unix process launch information
trace_workflow_agent	Trace messages generated by the workflow agent

Changing server tracing after start-up

Use the SET_OPTIONS administration method to turn tracing on or off after starting the server. The syntax is:

```
EXECUTE set_options
WITH option='option_name', "value"=true|false
```

or

```
apply, session, NULL, SET_OPTIONS, OPTION, S, option_name,
VALUE, B, T|F
```

Set the value argument to true to start tracing and to false to turn it off. You can use any option listed in [Table 12-23, page 543](#). You can also use the `last_sql_trace` option. This option is on by default when a server starts. You can use `SET_OPTIONS` to turn it off, but that is not recommended by Documentum Technical Support.

To turn off the `nettrace` option, disconnect all client sessions. After about one minute, the tracing turns off.

Examples of server tracing

The following example demonstrates full-text tracing. Turn on full-text tracing as follows:

```
API>apply, c, NULL, MODIFY_TRACE, SUBSYSTEM, S, fulltext, VALUE, S,
all
```

Here is the corresponding server log excerpt:

```
select: select all dm_workitem.r_object_id from dm_workitem_sp dm_workitem
where (dm_workitem.r_runtime_state in (0, 1) and
dm_workitem.r_auto_method_id!='0000000000000000') order by
dm_workitem.r_creation_date

select: select all dm_job.r_object_id, dm_job.a_last_invocation,
dm_job.a_last_completion, dm_job.a_special_app from dm_job_sp dm_job
where ( ( (dm_job.a_last_invocation >= to_date('02-JAN-0001', 'DD-MON-YYYY'))
and (dm_job.a_last_completion < to_date('02-JAN-0001', 'DD-MON-YYYY')) )
or (dm_job.a_special_app='agentexec') ) and (dm_job.i_has_folder = 1 and
dm_job.i_is_deleted = 0)

select: select all dm_job.r_object_id, dm_job.a_next_invocation from dm_job_sp
dm_job where ( ( (dm_job.run_now=1) or ( (dm_job.is_inactive=0) and
( (dm_job.a_next_invocation<= to_date('18/08/1998 14:15:15',
'DD/MM/YYYY HH24:MI:SS') and dm_job.a_next_invocation >=
to_date('02-JAN-0001', 'DD-MON-YYYY')) or (dm_job.a_next_continuation<=
to_date('18/08/1998 14:15:15', 'DD/MM/YYYY HH24:MI:SS') and
dm_job.a_next_continuation >= to_date('02-JAN-0001', 'DD-MON-YYYY')) )
and ( (dm_job.expiration_date> to_date('18/08/1998 14:15:15',
'DD/MM/YYYY HH24:MI:SS')) or (dm_job.expiration_date <
to_date('02-JAN-0001', 'DD-MON-YYYY')) ) and ( (dm_job.max_iterations=0)
or (dm_job.a_iterations<dm_job.max_iterations) ) ) ) )
and (dm_job.i_has_folder = 1 and dm_job.i_is_deleted = 0)
order by dm_job.a_next_invocation, dm_job.r_object_id
```

The following example demonstrates SQL tracing.

First, connect to a repository through the API and issue the following commands:

```
API> retrieve, c, dm_user
...
110007b680000100
API> apply, c, NULL, SET_OPTIONS, OPTION, S, sqltrace, VALUE, B, T
```

```

...
q0
API> retrieve,c,dm_server_config
...
3d0007b680000101

```

At this point, there is no SQL trace information in server log.

If you then start another session:

```

API> trace,c,11,,SQL_TRACE
...
OK
API> getmessage,c
...
[DM_API_I_SQL_TRACE_USAGE]info: "SQL Trace is currently (on).
The api command to turn on[/off] SQL trace is:
trace,<session>,1[/0],,SQL_TRACE"
API> query,c,select count(*) from dm_user
...
q0

```

Here is the corresponding server log excerpt:

```
select all count(*) from dm_user_sp dm_user
```

The following example demonstrates debug tracing.

Turn on debug tracing as follows:

```

API> apply,c,NULL,SET_OPTIONS,OPTION,S,debug,VALUE,B,T
...
q0
API> retrieve,c,dm_server_config
...
3d0007b680000101

```

Here is the corresponding server log excerpt:

```

<<<<: Apply_ext, Cost in micro secs: 41300000
Dispatching complete
Dispatching for connection: 0
>>>>: Apply_ext
<<<<: Apply_ext, Cost in micro secs: 1482000
Dispatching complete
Dispatching for connection: 0
>>>>: Next_ext
<<<<: Next_ext, Cost in micro secs: 70000
Dispatching complete
Dispatching for connection: 0
>>>>: ApplyForObject_ext
<<<<: ApplyForObject_ext, Cost in micro secs: 10000
Dispatching complete
Dispatching for connection: 0
>>>>: ApplyForObject_ext
<<<<: ApplyForObject_ext, Cost in micro secs: 381000
Dispatching complete
Dispatching for connection: 0
Change check called 46 times at Tue Aug 11 12:22:48 1998
>>>>: Apply_ext

```

```
<<<<: Apply_ext, Cost in micro secs: 11176000
Dispatching complete
Dispatching for connection: 0
```

Session tracing

The Trace method turns on session tracing at a specified severity level.

Trace levels define a minimum severity level for the trace messages sent to the session log file, stored in %DOCUMENTUM%\dba\log\hex_repository_id\username (\$DOCUMENTUM/dba/log/hex_repository_id/username), where *hex_repository_id* is the repository ID expressed as a hexadecimal value and *username* is the account under which the session is running. The server creates a separate session log for each new connection.

The syntax of the Trace method is:

```
dmAPIExec ("trace, session, severity_level[, apilogfile],
option|facility")
```

For a complete description of the Trace method, refer to [Trace, page 456](#), in the *Content Server API Reference Manual*.

Severity levels

[Table 12–24, page 546](#), lists valid trace level severity values and their corresponding meaning:

Table 12-24. Trace level severity values

Severity level	Meaning
0	No messages, tracing is turned off
1	Level 1 trace messages
2	Level 2 trace messages
3	Level 3 trace messages
4	Level 4 trace messages
8	Dump and Load object information
10	Timing information
11	Information messages

Any severity level value from 1 to 10 also turns on tracing to the `api.log` file. If you do not specify a facility when you use the Trace method to turn tracing on, the server turns on tracing for the API.

The severity levels are additive. For example, if you specify tracing level 10, you also receive all the other messages specified by the lower levels in addition to the timing information.

To stop tracing, issue the Trace method and specify zero or a negative number for the severity.

Tracing severity level 10

If you specify a level of 10, you receive information about how long each API method takes to execute. To illustrate, here are a few sample commands and the timing trace information they generate if the severity level is set to 10:

```
# Time0.00 sect 'OK'
create,c,dm_document
# Time0.024 sec '090007d080037572'
setfile,c,l,api.c,text
# Time1.924 sec 'OK'
save,c,l
# Time0.872 sec 'OK'
```

Tracing severity level 11

The severity level 11 returns information about the available trace facilities and options. Facilities are the session tracing options available through the Trace method, and options are the server tracing options available at start-up or through the Apply method.

For example, to see a list of all available trace facilities and options:

```
dmAPIExec("trace,c,11")
```

To get instructions on how to turn on the specific tracing facility:

```
dmAPIExec("trace,c,11,,facility")
```

You can use severity level 11 in debugging if you are having a problem with a load operation. Generally, load operations return status messages to the server log file after every 100 objects are loaded. If you are having a problem with a load, you can set the trace level to 11 and the operation will return status messages for each object loaded. However, this does severely impact the performance of the load operation.

Examples of session tracing

The following example demonstrates load tracing. Turn on load tracing as follows:

```
API> trace,c,10,,DM_LOAD
...
OK
```

Then start a load operation:

```
API> create,c,dm_load_record
...
310007b680000101
API> set,c,l,file_name
SET> c:\dump_test
...
OK
API> save,c,l
...
OK
```

Here is the corresponding server log excerpt:

```
Fri Aug 14 16:02:27 1998 569000 [DM_LOAD_I_PROGRESS]info:
"For load object 310007b680000101 in phase 2, processed 54 of 66 objects;
last object processed was 090007b680000238"
Fri Aug 14 16:02:27 1998 710000 [DM_LOAD_I_PROGRESS]info:
"For load object 310007b680000101 in phase 2, processed 55 of 66 objects;
last object processed was 060007b680000118"
Fri Aug 14 16:02:27 1998 720000 [DM_LOAD_I_PROGRESS]info:
"For load object 310007b680000101 in phase 2, processed 56 of 66 objects;
last object processed was 270007b680000165"
```

The following example demonstrates query tracing. Turn on query tracing:

```
API> trace,c,8,,DM_QUERY
...
OK
```

Then issue a query:

```
API> query,c,select user_os_name from dm_user where user_name='zhan'
...
q0
```

Here is the corresponding server log:

```
Fri Aug 14 15:31:29 1998 608000 [DM_QUERY_T_SELECT_COMPLETE]info:
"SELECT statement semantic checking and setup is complete."
Fri Aug 14 15:32:20 1998 942000 [DM_QUERY_T_SYNTAX_BEGIN]info:
"Begin syntactic parse (call yacc)."
```

```
Fri Aug 14 15:32:20 1998 942000 [DM_QUERY_T_SYNTAX_COMPLETE]info:
"Syntactic parse is complete."
Fri Aug 14 15:32:20 1998 942000 [DM_QUERY_T_SELECT_BEGIN]info:
"Begin SELECT statement."
Fri Aug 14 15:32:20 1998 942000 [DM_QUERY_T_SQL_SELECT]info:
"SELECT statement generated by the query is:
select all dm_user.user_os_name from dm_user_sp dm_user where
(dm_user.user_name='zhan')."
```

```
Begin Database processing."  
Fri Aug 14 15:32:22 1998 434000 [DM_QUERY_T_SELECT_COMPLETE]info:  
"SELECT statement semantic checking and setup is complete."
```

Client tracing

When using the Apply method to turn on tracing, the client can use the `save_results` option to get the session log to the client. If you do not use the `save_results` option, the client can retrieve the path of the session log.

For client log files, the *severity_level* specified in the Trace method must be greater than 0 to turn on tracing, but has no other effect.

Consistency Checks

This appendix describes the consistency checks executed by the Consistency Checker administrative tool. The checks are sorted into tables that describe the checks on a particular area of the repository. The tables are:

- [User and group checks, page 552](#)
- [ACL checks, page 553](#)
- [SysObject checks, page 554](#)
- [Folder and cabinet checks, page 555](#)
- [Document checks, page 556](#)
- [Content object checks, page 556](#)
- [Workflow checks, page 557](#)
- [Object type checks, page 558](#)
- [Data dictionary checks, page 558](#)
- [Lifecycle checks, page 559](#)
- [Object type index checks, page 560](#)
- [Method object consistency checks, page 561](#)

Each table includes the following information:

- Consistency check number

Each consistency check has a unique number. When an inconsistency is reported, the report includes the consistency check number and some information about the particular inconsistency. For example:

```
WARNING CC-0001: User docu does not have a default group
```

- Description
- Severity level

The consistency checker script reports inconsistencies with one of two severity values: Warning or Error.

- A check produces a warning for a referential integrity inconsistency. For example, you would see a warning if a check found a document referencing an author who no longer exists in the repository. You should fix such problems, but they do not threaten repository operations.
- A check produces an error for inconsistencies requiring immediate resolution. These include object corruption problems, such as missing `_r` table entries or missing entries in a `dmi_object_type` table, and type corruption.

User and group checks

These consistency checks apply to users and groups.

Table A-1. Consistency checks for users and groups

Consistency check number	Consistency check description	Severity
CC-0001	Check for users who belong to a group that does not exist	Warning
CC-0002	Check for users who belong to groups that are not in <code>dm_user</code>	Warning
CC-0003	Check for users who are not listed in <code>dmi_object_type</code>	Error
CC-0004	Check for groups that are not listed in <code>dmi_object_type</code>	Error
CC-0005	Check for groups that belong to groups that do not exist	Warning
CC-0006	Check for groups belonging to supergroups that do not exist	Warning
CC-0081	Check for groups with disconnected super groups	Warning
CC-0082	Check for groups with disconnected subgroups	Warning

ACL checks

These consistency checks apply to ACLs.

Table A-2. Consistency checks for ACLs

Consistency check number	Consistency check description	Severity
CC-0007	Check for ACLs containing users who do not exist in the repository	Warning
CC-0008	Check for ACLs which are missing dm_acl_r entries	Error
CC-0009	Check for sysobjects whose acl_domain is set to a user who does not exist in the repository	Warning
CC-0010	Check for sysobjects that belong to users who do not exist in the repository	Warning
CC-0011	Check for sysobjects that are set to ACLs that do not exist	Warning
CC-0012	Check for ACL objects with missing dm_acl_s entry	Error
CC-0013	Check for ACL objects with r_accessor_permit values that are missing r_accessor_name values	Error
CC-0014	Check for ACL objects with r_accessor_name values that are missing r_accessor_permit values	Error
CC-0015	Check for ACL objects with r_is_group values that are missing r_accessor_permit values	Error

Consistency check number	Consistency check description	Severity
CC-0016	Check for ACL objects with r_is_group values that are missing r_accessor_name values	Error
CC-0017	Check for ACL objects with r_accessor_name values that are missing r_is_group values	Error
CC-0018	Check for ACL objects with r_accessor_permit values that are missing r_is_group values	Error

SysObject checks

These consistency checks apply to SysObjects.

Table A-3. Consistency checks for SysObjects

Consistency check number	Consistency check description	Severity
CC-0019	Check for sysobjects that are not referenced in dmi_object_type	Error
CC-0020	Check for sysobjects that point to non-existent content	Warning
CC-0021	Check for sysobjects that are linked to non-existent folders	Warning
CC-0022	Check for sysobjects that are linked to non-existent primary cabinets	Warning
CC-0023	Check for sysobjects that reference non-existent i_chronicle_id	Warning

Consistency check number	Consistency check description	Severity
CC-0024	Check for sysobjects that reference non-existent i_antecedent_id	Warning
CC-0025	Check for sysobjects with dm_sysobject_s entry but missing dm_sysobject_r entries	Error
CC-0026	Check for sysobjects with dm_sysobject_r entries but missing dm_sysobject_s entries	Error

Folder and cabinet checks

These consistency checks apply to folders and cabinets.

Table A-4. Consistency checks for folders and cabinets

Consistency check number	Consistency check description	Severity
CC-0027	Check for folders with missing dm_folder_r entries	Error
CC-0028	Check for folders that are referenced in dm_folder_r but not in dm_folder_s	Error
CC-0029	Check for dm_folder objects that are not referenced in dmi_object_type	Error
CC-0030	Check for dm_folder objects that are missing dm_sysobject entries	Error
CC-0031	Check for folders with non-existent ancestor_id	Warning

Consistency check number	Consistency check description	Severity
CC-0032	Check for cabinet objects that are missing dm_folder_r table entries	Error
CC-0033	Check that cabinet objects are not referenced in dmi_object_type	Error
CC-0034	Check for folder objects that are missing dm_sysobject_r entries	Error
CC-0035	Check for folder objects with null r_folder_path	Error

Document checks

These consistency checks apply to documents.

Table A-5. Consistency checks for documents

Consistency check number	Consistency check description	Severity
CC-0036	Check for documents with a dm_sysobject_s entry but no dm_document_s entry	Error
CC-0037	Check for documents with missing dm_sysobject_s entries	Error
CC-0038	Check for documents with missing dmi_object_type entry	Error

Content object checks

These consistency checks apply to content objects.

Table A-6. Consistency checks for content objects

Consistency check number	Consistency check description	Severity
CC-0039	Check for content objects referencing non-existent parents	Warning
CC-0040	Check for content with invalid storage_id	Warning
CC-0041	Check for content objects with non-existent format	Warning

Workflow checks

These consistency checks apply to workflows.

Table A-7. Consistency checks for workflows

Consistency check number	Consistency check description	Severity
CC-0042	Check for dmi_queue_item objects with non-existent queued objects	Warning
CC-0043	Check for dmi_workitem objects that reference non-existent dm_workflow objects	Warning
CC-0044	Check for workflow objects with missing dm_workflow_s entry	Error
CC-0045	Check for dmi_package objects with missing dmi_package_s entries	Error
CC-0046	Check for dmi_package objects that reference non-existent dm_workflow objects	Warning

Consistency check number	Consistency check description	Severity
CC-0047	Check for workflow objects with non-existent r_component_id	Warning
CC-0048	Check for dmi_workitem objects with missing dmi_workitem_s entry.	Error

Object type checks

These consistency checks apply to object types.

Table A-8. Consistency checks for object types

Consistency check number	Consistency check description	Severity
CC-0049	Check whether any dm_type objects reference non-existent dmi_type_info objects	Error
CC-0050	Check whether any dmi_type_info objects reference non-existent dm_type objects	Error
CC-0051	Check whether any types have corrupted attribute positions	Error
CC-0052	Check whether any types have invalid attribute counts	Error

Data dictionary checks

These checks apply to the data dictionary.

Table A-9. Consistency checks for the data dictionary

Consistency check number	Consistency check description	Severity
CC-0053	Check whether any duplicate dmi_dd_attr_info objects exist.	Error
CC-0054	Check whether any duplicate dmi_dd_type_info objects exist	Error
CC-0055	Check whether any dmi_dd_attr_info objects are missing an entry in dmi_dd_common_info_s	Error
CC-0056	Check whether any dmi_dd_type_info objects are missing an entry in dmi_dd_common_info_s	Error
CC-0057	Check whether any dmi_dd_attr_info objects are missing an entry in dmi_dd_attr_info_s	Error
CC-0058	Check whether any dmi_dd_type_info objects are missing an entry in dmi_dd_type_info_s	Error
CC-0078	Check whether any data dictionary objects reference non-existent default policy objects	Error

Lifecycle checks

These checks apply to document lifecycles.

Table A-10. Consistency checks for lifecycles

Consistency check number	Consistency check description	Severity
CC-0059	Check for any dm_sysobject objects that reference non-existent dm_policy objects	Warning
CC-0060	Check for any policy objects that reference non-existent types in included_type	Warning
CC-0061	Check for any policy objects with missing dm_sysobject_s entries	Error
CC-0062	Check for any policy objects with missing dm_sysobject_r entries	Error
CC-0063	Check for any policy objects with missing dm_policy_r entries	Error
CC-0064	Check for any policy objects with missing dm_policy_s entries	Error

Object type index checks

These consistency checks apply to object type indexes.

Table A-11. Consistency checks for object type indexes

Consistency check number	Consistency check description	Severity
CC-0073	Check for dmi_index objects that point to non-existent types	Warning
CC-0074	Check for types with non-existent dmi_index object for _s table	Warning

Consistency check number	Consistency check description	Severity
CC-0075	Check for types with non-existent dmi_index object for _r table	Warning
CC-0076	Check for indexes with invalid attribute positions	Warning

Method object consistency checks

These consistency checks apply to method objects.

Table A-12. Consistency checks for method objects

Consistency check number	Consistency check description	Severity
CC-0077	Check for methods using jview rather than Java	Warning

IAPI and IDQL

This appendix describes the Interactive API (IAPI) and Interactive DQL (IDQL) utilities. These utilities are useful if you want to test individual method calls or short scripts or perform a short, small tasks in the repository. The appendix includes the following topics:

- [Using IAPI, page 563](#)
- [Using IDQL, page 569](#)

Using IAPI

The IAPI utility is a tool that you can use to send individual method calls to the server. This utility is included and installed with Content Server. It is found in %DM_HOME%\bin (\$DM_HOME/bin). Copy the utility's executable (iapi32.exe or iapi) from the bin directory to a directory you can access or copy it to your local machine.

If you copy it to your local machine, you must also copy the client library (DMCL) to your local machine if it is not already there. If you have any of Documentum's desktop clients or Developer Studio installed on your machine, then you have a copy of the DMCL. However, the DMCL is not installed with the Web-based clients.

Starting IAPI

You can start the utility from the operating system prompt or an icon. To start the utility from the operating system prompt, use the following syntax:

On Windows:

```
iapi32 [-Username|-ENV_CONNECT_USER_NAME]
[-Pos_password|-ENV_CONNECT_PASSWORD] [-Llogfile] [-X][-E]
[-V-][-Q] repository|-ENV_CONNECT_DOCBASE_NAME [-Rinputfile]
[-Ftrace_level] [-Sinit_level] [-Iescape_character] [-zZ]
[-Winteger] [-Ksecure_mode]
```

On UNIX:

```
iapi [-Username|-ENV_CONNECT_USER_NAME]
[-Pos_password|-ENV_CONNECT_PASSWORD] [-Llogfile] [-X] [-E]
[-V-] [-Q] repository|-ENV_CONNECT_DOCBASE_NAME [-Rinputfile]
[-Ftrace_level] [-Sinit_level] [-Iescape_character] [-zZ]
[-Winteger] [-Ksecure_mode]
```

The order of the arguments on the command line is not important.

If you created an icon for the utility, double-click the icon and enter your name and password. To invoke the utility with additional command line arguments, use the icon's Properties to add the arguments to the command line before you double-click the icon.

[Table B-1, page 564](#), describes the command line arguments.

Table B-1. IAPI command line arguments

Argument	Description
-Username	Identifies the user who is starting the session. You can include the user name on the command line with the -U flag or you can use the -ENV_CONNECT_USER_NAME argument, which directs the system to look for the user name in the environment variable DM_USER_NAME.
or	
-ENV_CONNECT_USER_NAME	
-Pos_password	Identifies the user's operating system password. You can include the password on the command line with the -P flag, or you can use the -ENV_CONNECT_PASSWORD argument, which directs the system to look for the password in the environment variable DM_PASSWORD.
or	
-ENV_CONNECT_PASSWORD	
	If the user doesn't specify a password, the utility prompts for the password.
-Llogfile	Directs the server to start a log file for the session. You must include the file name with the argument. You can specify a full path.
-X	Allows prompt for domain.
-E	Turns on echo.

Argument	Description
-V-	<p>Turns verbose mode off. The utility runs in verbose mode by default. If you turn verbose mode off, the system does not display the IAPI prompt, error messages, or any responses to your commands (return values or status messages).</p> <p>You can also change modes from the IAPI prompt (refer to IAPI commands, page 566).</p>
-Q	<p>Directs the utility to provide a quick connection test. The utility attempts to connect and exits with a status of zero if successful or non-zero if not. It provides error messages if the attempt is not successful.</p>
<p><i>repository</i></p> <p>or</p> <p>-ENV_CONNECT_DOCBASE_NAME</p>	<p>Identifies the repository. You can include the repository name on the command line, or you can use the -ENV_CONNECT_DOCBASE_NAME argument, which directs the system to look for the repository name in the environment variable DM_DOCBASE_NAME.</p>
-R <inputfile< td=""> <td> <p>Identifies an input file containing API methods.</p> </td> </inputfile<>	<p>Identifies an input file containing API methods.</p>
-Ftrace_level	<p>Turns tracing on for the session. <i>trace_level</i> can be any of the valid trace levels described in the Trace method description.</p>
-Sinit_level	<p>Defines the connection level established when you start the API session. Valid init levels and their meanings are:</p> <p>api, which opens an API session. User must issue an explicit Connect to start a session.</p> <p>connect, meaning undetermined</p> <p>standard, which opens a session</p>
-Iescape_character	<p>Defines an escape character for certain symbols such as a forward slash (/).</p>

Argument	Description
-zZ	Specifies Windows NT Unified Logon. This option overrides any user name and password specified on the command line.
-Winteger or -winteger	Sets the maximum column width when displaying results returned from a query entered with a ? command. <i>integer</i> is interpreted as number of bytes. The default is 2000 bytes.
-Ksecure_mode	Any value which exceeds the maximum row width is truncated when displayed. Defines the type of connection you want to establish. Valid values are: <ul style="list-style-type: none"> • secure • native • try_secure_first • try_native_first The default is native. For an explanation of the values, refer to Requesting a native or secure connection, page 168 . For an explanation of how the values interact with the server's secure connection default, refer to Connect, page 155 of the <i>Content Server API Reference Manual</i> .

When you issue the `iapi` command, the system returns a status message that tells you it is connecting you to the specified database. This is followed by a message giving you the session ID for your session and then the utility's prompt (API>).

IAPI commands

The IAPI commands affect how the IAPI utility functions. These commands do not affect the repository. The utility accepts the commands listed in [Table B-2, page 567](#).

Table B-2. IAPI commands

Command	Description
<i>\$logfile</i>	Outputs to the specified file
<i>@scriptfile</i>	Reads input from the specified file
<i>%scriptfile</i>	Outputs a script
<i>-V[+ -]</i>	Turns verbose mode on (+) or off (-). By default, verbose mode is on. If you turn verbose mode off, the system does not display the IAPI prompt, error messages, or any responses to your commands (return values or status messages).
<i>-Winteger</i> or <i>-winteger</i>	Sets the maximum column width when displaying results returned from a query entered with a <i>? </i> command. <i>integer</i> is interpreted in bytes. The default is 2000 bytes. Setting this affects the results display for all methods issued after this command.
<i>?</i>	Allows you to enter a query or collection ID and formats the output as a table.

The *?* command allows you to enter a DQL SELECT statement in IAPI. When used with a SELECT statement, the command is the equivalent of a Readquery method. The syntax is:

```
API>?,c,select_query_string
```

For example:

```
API>?,c,select r_object_id, object_name from dm_document
```

The results are formatted in a table-like block. For example:

```
r_object_idobject_name
-----
090015c38000013Bfoo.txt
090015c3800002e4spam.html
090015c3800001c2fah.txt
```

You can also use the *?* command to retrieve a collection's content in one block. For example, suppose you issue a readquery method, which returns a collection:

```
API>readquery,c,select r_object_id, object_name from dm_document
...
API>q0
```

Instead of using a Next method to retrieve each row of the collection, you can use *?* to retrieve them all at once:

```
API>?, c, q0
...
r_object_idobject_name
-----
090015c38000013Bfoo.txt
090015c3800002e4spam.html
090015c3800001c2fah.txt
```

Entering method calls

Call methods directly from the IAPI prompt by typing the method name and its arguments as a continuous string, with commas separating the parts. For example, the following command creates a new folder:

```
API> create, s0, dm_folder
```

If any parameter value contains a comma, you must enclose that parameter in single quotes. For example, the following command creates a new document and assigns it a content file:

```
API> create, s0, dm_document
API> setfile, s0, last, '/home/janed/temp.doc, 1', text
```

The name of the content file is in single quotes because it contains a comma.

When the method completes, the utility displays the return value. For example, the Create method returns the object ID of the newly created object. So if you issue the Create command shown in the previous example, the utility displays the object ID of the new folder:

```
API> create, s0, dm_folder
. . .
0b6385F20000561B
```

When you execute a method that assigns a value to an attribute, you do not specify the value as part of the command syntax. Instead, IAPI prompts you for that value. For example, the following exchange sets the name of the folder object created in the previous example:

```
API> set, s0, last, object_name
SET> cake_recipes
. . .
OK
```

The keyword `last` refers to the last created, fetched, checked in/out, or retrieved object ID (in this case, the folder ID).

Entering comments

Enter a comment in an IAPI session by preceding it with the pound sign (#). This is useful when you want to use scripts with IAPI.

Quitting an IAPI session

To exit from IAPI, enter quit at the IAPI prompt:

```
IAPI> quit
```

Using IDQL

The IDQL utility is an interactive tool that lets you enter ad hoc DQL queries against a repository. IDQL is also a useful as a tool for testing and other tasks that support an application or installation because it allows you to run scripts and batch files.

IDQL is included and installed with Content Server. It is found in %DM_HOME%\bin (\$DM_HOME/bin). Copy the utility's executable (idql32.exe or idql) from the bin directory to a directory you can access or copy it to your local machine.

If you copy it to your local machine, you must also copy the client library (DMCL) to your local machine if it is not already there. If you have any of Documentum's desktop clients or Developer Studio installed on your machine, then you have a copy of the DMCL also. However, the DMCL is not installed with the Web-based clients.

Starting IDQL

You can start the utility from the operating system prompt or an icon. To start the utility from the operating system prompt, use the following syntax:

On Windows:

```
idql32 [-n] [-wcolumnwidth]
[-Username| -ENV_CONNECT_USER_NAME]
[-Pos_password| -ENV_CONNECT_PASSWORD] [-Ddomain]
repository| -ENV_CONNECT_DOCBASE_NAME [-Rinput_file]
[-X] [-E] [-Ltrace_level] [-zZ] [-Winteger] [-Ksecure_mode]
```

On UNIX:

```
idql [-n] [-wcolumnwidth]
[-Username| -ENV_CONNECT_USER_NAME]
[-Pos_password| -ENV_CONNECT_PASSWORD] [-Ddomain]
```

```
repository| -ENV_CONNECT_DOCBASE_NAME [-Rinput_file]
[-X] [-E] [-Ltrace_level] [-zZ] [-Winteger] [-Ksecure_mode]
```

The order of the arguments on the command line is not important.

If you created an icon for the utility, double-click the icon and enter your name and password. To invoke the utility with additional command line arguments, use the icon's Properties to add the arguments to the command line before you double-click the icon.

Table B-3, page 570, describes the command line arguments.

Table B-3. IDQL command line arguments

Argument	Description
-n	Removes numbering and the prompt symbol from input files. This argument is used primarily for scripts.
-wcolumnwidth	Sets the screen width for output. The default width is 80 characters.
-Username	Identifies the user who is starting the session. You can include the user name on the command line with the -U flag, or you can use the -ENV_CONNECT_USER_NAME argument, which directs the system to look for the user name in the environment variable DM_USER_NAME.
or	
-ENV_CONNECT_USER_NAME	
-Pos_password	Identifies the user's operating system password. You can include the password on the command line with the -P flag, or you can use the -ENV_CONNECT_PASSWORD argument, which directs the system to look for the password in the environment variable DM_PASSWORD.
or	
-ENV_CONNECT_PASSWORD	
	If the user does not specify a password, the utility prompts for the password. If the user enters the -P flag without the password, IDQL uses the default password, NULL. Passwords are case sensitive.
-Ddomain	Identifies the user's domain.

Argument	Description
<i>repository</i>	Identifies the repository. You can include the repository name on the command line, or you can use the <code>-ENV_CONNECT_DOCBASE_NAME</code> argument, which directs the system to look for the repository name in the environment variable <code>DM_DOCBASE_NAME</code> .
or	
<code>-ENV_CONNECT_DOCBASE_NAME</code>	
<code>-Rinputfile</code>	Specifies the name of an input file containing DQL queries. This is an optional argument.
<code>-X</code>	Allows prompt for domain.
<code>-E</code>	Turns on echo.
<code>-Ltrace_level</code>	Turns tracing on for the session. <i>trace_level</i> can be any of the valid trace levels described in the Trace method description.
<code>-zZ</code>	Specifies Windows NT Unified Logon. This option overrides any user name and password specified on the command line.
<code>-Winteger</code>	Sets the maximum column width when displaying results. <i>integer</i> is interpreted as number of bytes. The default is 2000 bytes.
or	
<code>-winteger</code>	Any value which exceeds the maximum row width is truncated when displayed.
<code>-Ksecure_mode</code>	Defines the type of connection you want to establish.
	Valid values are:
	<ul style="list-style-type: none"> • secure • native • try_secure_first • try_native_first
	The default is native. For an explanation of the values, refer to Requesting a native or secure connection, page 168 . For an explanation of how the values interact with the server's secure connection default, refer to Connect, page 155 , of the <i>Content Server API Reference Manual</i> .

After you start IDQL, the utility returns with its prompt. The IDQL utility has a numerical prompt that begins with 1 and increments each time you press the carriage return or Enter key to move to a new line on your display.

The IDQL commands

IDQL recognizes the commands listed in [Table B-4, page 572](#).

Table B-4. IDQL commands

Command	Description
go	Executes a DQL statement and clears the query buffer.
clear	Clears the query buffer.
quit	Exits the utility.
describe	Provides a description of a specified object type or registered table. The syntax formats for this command are: <pre>describe [type] type_name describe table table_name</pre>
@ <i>scriptfile</i>	Executes the specified script.
shutdown	Issues a Shutdown method that shuts down the Content Server gracefully. You must be a superuser or system administrator to use this command. When shutdown is complete, the utility exits.
trace	Issues a Trace method to turn on tracing for the current session. By default, tracing is set to level 5. (Refer to the <i>Content Server API Reference Manual</i> for information about the Trace, page 456, method .)
- <i>Winteger</i>	Sets the maximum column width when displaying results. <i>integer</i> is interpreted in bytes. The default is 2000 bytes.
or	
- <i>winteger</i>	Setting this affects the results display for all statements issued after this command.

Each command must be entered on a separate line. A command cannot appear on the same line as a query, nor can two or more commands appear on one line.

Entering queries

Enter a query by typing it at the IDQL prompt. The following IDQL session shows two queries typed at the prompt:

```
1> BEGIN TRAN
2> go
3> SELECT *
4> FROM Engineering
5> WHERE Engineer LIKE 'Smith, %'
6> go
```

The query processor interprets semicolons (;) as statement terminators and ignores everything that follows the semicolon on the line.

When you enter a query, IDQL places the query in a buffer. You can then execute the query or clear the buffer.

The `go` command executes a query. Each `go` command can execute only one query. You cannot execute multiple queries with one `go` command.

There are two ways to execute queries in a batch using IDQL:

- Use the input file option in the IDQL command line.
- Use the `@scriptfile` command.

The input file is a file that contains queries. You must place a `go` command after each query in the file, and you must terminate the file with a `quit` command.

The results of each query are returned to standard output by default.

The `@scriptfile` command executes a specified script, which can include DQL queries. Like an input file, queries in the script must be separated by `go` commands. You are not required to use a `quit` command to terminate a script executed with `@scriptfile`.

Clearing the buffer

The following excerpt enters a query and then clears it from the buffer:

```
1> SELECT *
2> FROM Engineering
3> WHERE Engineer LIKE 'Smith, %'
4> clear
```

Entering comments

To enter a comment in an IDQL session, start the comment line with `\ \` or `--`.

Stopping IDQL

To close an IDQL session, enter the `quit` command at the IDQL prompt.

Archiving scripts

This appendix contains the text of the scripts provided by Documentum to support the archiving functionality. You must edit these scripts before running to them to localize them for your installation.

The dql Script

```
#
# dm_archive.dql - initialize methods for the dmarchive
# utility.
# This file was created by user, roger,
# on Thu Nov  3 21:26:00 PST 1994 using
# the utility ./tools/dm_archive.gen
#
# This script creates the methods used by the archive utility
# "dmarchive".
# These methods will be called if they exist and are intended
# to allow the archive facility to be customized to move the
# archive dumpfiles to an offline storage medium until needed
# by restore actions.
#
# post_archive - Called after each archive action, its
# intention is to allow the created archive dumpfile to be
# moved from the archive directory to offline storage
# location.
#
#The method invokes a script with two arguments:
#
# $1 - The name of the new archive dumpfile (path included)
# $2 - The number of objects dumped by the archive operation.
#
# pre_restore - Called prior to each restore action, its
# intention is to
# allow the requested archive dumpfile to be restored to
# the archive directory from any offline storage area it
# may have been moved to by the post_archive method.
#
# The method invokes a script with one arguments:
#
```

```
# $1 - The name of requested archive dumpfile, including the
# filesystem path. It must be restored to its original
# filesystem directory.
#
# post_restore - Called after each restore action, its
# intention is to allow the archive dumpfile to be removed
# from the archive directory since its use is now complete.
#
# The method invokes a script with one arguments:
#
# $1 - The name of requested archive dumpfile, including the
# filesystem path. It must be restored to its original
# filesystem directory.
#

create dm_method object
set object_name = 'post_archive',
set method_verb = '/u31/docbase/roger/dba/post_archive',
set timeout_min = 30,
set timeout_max = 1000,
set timeout_default = 1000,
set launch_direct = false,
set launch_async = false,
set trace_launch = false,
set run_as_server = true
go

create dm_method object
set object_name = 'pre_restore',
set method_verb = '/u31/docbase/roger/dba/pre_restore',
set timeout_min = 30,
set timeout_max = 1000,
set timeout_default = 1000,
set launch_direct = false,
set launch_async = false,
set trace_launch = false,
set run_as_server = true
go

create dm_method object
set object_name = 'post_restore',
set method_verb = '/u31/docbase/roger/dba/post_restore',
set timeout_min = 30,
set timeout_max = 1000,
set timeout_default = 1000,
set launch_direct = false,
set launch_async = false,
set trace_launch = false,
set run_as_server = true
go
```

The dm_archive_start script

This script starts a dmarchive process.

```

#!/bin/sh
# dm_archive_start - start archive program for repository
#
# This file was created by user, roger,
# on Wed Nov  2 16:06:47 PST 1994 using
# the utility dm_setup_archive
#
# This script should normally be run by the repository operator
# user.
#
# This script may be modified if desired to alter the archive
# behavior by adding the following flags to the startup
# options below:
#
# -U<username>- Run as specified user (eg. repository operator
# name)
#
# -queue_name <name>- Specify alternate inbox queue on
# which requests are made.
#
# -do_archive_evnts - Specify that program should only
# handle archive requests (not restore requests). Implies
# that restore requests will be handled by another method or a
# different invocation of this program.
#
# -do_restore_events - Specify that program should only
# handle restore requests (not archive requests). Implies
# that archive requests will be handled by another method
# or a different invocation of this program.
#
#
logfile=/u31/docbase/roger/dba/log/dmarchive.rogbase.log
dmarchive docbase_name -P -archive_dir /tmp/archive
-verbose >> $logfile &

```

The post_archive script

This script is meant to be run after an archive request is completed, to remove the dump file from the archive directory and place it in permanent storage.

The Windows version:

```

@echo off
rem
rem Id$
rem
rem post_archive - Called after each archive action,
rem its intention is to
rem allow the created archive dumpfile to be moved from the
rem archive directory to offline storage location.
rem
rem The post_archive script is invoked with two arguments:
rem

```

```
rem %1 - The name of the new archive dumpfile (path included)
rem %2 - The number of objects dumped by the archive operation.
rem      Primarily provided so that the dumpfile can be
rem      destroyed rather than saved if the object count is
rem      zero, which occurs when an archive is given which
rem      specifies only documents already archived.
rem
rem
rem This script is provided as a mechanism for automatically moving
rem archive dumpfiles to an offline storage area
rem following an archive invocation.
rem
rem Each archive operation produces an archive dumpfile in the
rem Archive Staging Directory and then removes the archived
rem document content from the repository storage areas.
rem The archive dumpfile must be saved as it is the only
rem mechanism for restoring the removed content if the documents are
rem needed sometime later.
rem
rem
rem It is suggested that the archive dumpfiles be periodically moved
rem out of the Archive Staging Directory and placed
rem on some offline storage media such as a tapes or optical devices.
rem
rem
rem This can be done by periodically backing up the Archive Staging
rem Directory to tape, and then removing the backed up archive
rem dumpfiles. On NT, here is a BackUp utility under the Administrative
rem Tools program folder. Alternatively, this can be done via a
rem third-party backup software that provides a command
rem line interface.
rem
rem NOTE THAT ONCE A DUMPFILe IS REMOVED from the Archive Staging
rem Directory, procedures must be put in place to retrieve
rem that dumpfile if a restore request is made on an archived
rem document. The restore procedure requires
rem that the dumpfile be placed back in the Archive Staging Directory in
rem order to re-load the archived content. rem See the pre_restore script
rem for more information on these requirements.
rem
rem
rem The archive methods (post_archive, pre_restore, and post_restore
rem scripts) are called as a mechanism for automating the process
rem of moving archive dumpfiles in and out of the staging area.
rem
rem
rem The post_archive method is called after each archive operation
rem and allows for the newly-created dumpfile to be
rem automatically moved to the offline storage device.
rem A simple example of this would be to add the shell
rem commands at the end of this script to copy the new dumpfile to a
rem backup device:
rem
rem
rem
rem echo ": post_archive: Saving %filename%" >> \temp\archive.log
rem      ... backup command to transfer dumpfile to a tape or another
```

```

rem      disk ...
rem      if successful erase %filename%
rem
rem
rem
rem      Default post_archive implementation:
rem
rem      This default script does not attempt to move dumpfiles out of the
rem      staging directory, but it does check for and remove empty
rem      archive files.
rem
rem
rem      set filename="%1"
rem      set object_count="%2"
rem
rem      If no objects were dumped in the specified archive operation, then
rem      delete the created dumpfile rather than saving it. Unused
rem      dumpfiles like this are created if an archive request
rem      specifies only documents which are already fully archived.
rem
rem
rem      This is often done purposefully to move offline a set of previously
rem      archived documents which have since been restored. The
rem      archive request takes them offline, but does not actually
rem      dump the documents since they have already been archived.
rem
rem
rem      if %object_count% NOT = "0" goto the_end
rem      echo ": post_archive: Deleting unneeded %filename%" >> \temp\archive.log
rem      erase /f /q %filename%
rem      exit
rem      the_end:
rem      echo ": post_archive: Called with dumpfile %filename%" >>
rem      \temp\archive.log
rem      exit

```

The UNIX version:

```

#!/bin/sh
#$Id: post_archive,v 1.1 1994/11/03 00:29:47 roger Exp $
#
# post_archive - Called after each archive action, its intention is to allow the
# created archive dumpfile to be moved from the
# archive directory to offline storage location.
#
# The post_archive script is invoked with two arguments:
#
# $1 - The name of the new archive dumpfile (path included)
# $2 - The number of objects dumped by the archive
# operation.
# Primarily provided so that the dumpfile can be destroyed
# rather than saved if the object count is zero, which occurs
# when an archive is given which specifies only documents
# already archived.
#
#
# This script is provided as a mechanism for automatically

```

```
# moving archive dumpfiles to an offline storage area
# following an archive invocation.
#
# Each archive operation produces an archive dumpfile in
# the Archive Staging Directory and then removes the archived
# document content from the repository storage areas. The
# archive dumpfile must be saved as it is the only mechanism
# for restoring the removed content if the documents are
# needed sometime later.
#
# It is suggested that the archive dumpfiles be
# periodically moved out of the Archive Staging Directory and
# placed on some offline storage media such as a tapes or
# optical devices.
#
# This can be done by periodically backing up the Archive
# Staging Directory to tape via a standard unix utility
# (tar, cpio ..) or your favorite third-party backup
# software, and then removing the backed up archive
# dumpfiles.
#
# NOTE THAT ONCE A DUMPFILe IS REMOVED from the Archive
# Staging Directory, procedures must be put in place to
# retrieve that dumpfile if a restore request is made on an
# archived document. The restore procedure requires that the
# dumpfile be placed back in the Archive Staging Directory in
# order to re-load the archived content. See the pre_restore
# script for more information on these requirements.
#
# The archive methods (post_archive, pre_restore, and
# post_restore scripts) are called as a mechanism for
# automating the process of moving archive dumpfiles in and
# out of the staging area.
#
#
# The post_archive method is called after each archive
# operation and allows for the newly-created dumpfile to be
# automatically moved to the offline storage device. A
# simple example of this would be to add the shell commands
# at the end of this script to copy the new dumpfile to a tape
# device:
#
#   echo "`date`: post_archive: Saving $filename" >> /tmp
# /archive.log
#   tar -rf /dev/rmt0 $filename
#   status=$?
#   if [ $status = 0 ]
#   then
#     rm $filename
#   fi
#
#####
#
# Default post_archive implementation:
#
# This default script does not attempt to move dumpfiles
```

```

# out of the staging directory, but it does check for and
# remove empty archive files.
#
#####
filename="$1"
object_count="$2"
#
# If no objects were dumped in the specified archive
# operation, then delete the created dumpfile rather than
# saving it.  Unused dumpfiles like this are created if an
# archive request specifies only documents which are already
# fully archived.
#
# This is often done purposefully to move offline a set of
# previously archived documents which have since been
# restored.  The archive request takes them offline, but does
# not actually dump the documents since they have already
# been archived.
#
if [ $object_count = "0" ]
then
    echo "`date`: post_archive: Deleting unneeded $filename" >> /tmp/archive.log
    rm $filename
    exit 0
fi

echo "`date`: post_archive: Called with dumpfile $filename" >> /tmp/archive.log
exit 0

```

The pre_restore script

The Windows version:

```

@echo off
rem $Id: pre_restore.bat,v 2.1 1995/11/13 23:27:56 kinnard Exp $
rem
rem pre_restore - Called prior to each restore action, its
rem intention is to allow the requested archive dumpfile to be
rem restored to the archive directory from any offline storage
rem area it may have been moved to by the post_archive method.
rem The pre_restore script is invoked with one argument:
rem
rem 1 - The name of requested archive dumpfile, including
rem the filesystem path. It must be restored to its original
rem filesystem directory.
rem
rem
rem This script is provided as a mechanism for automatically restoring
rem archive dumpfiles from their offline storage area
rem so that their archived content can be retrieved by a restore
rem operation.
rem

```

```
rem Each archive operation produces an archive dumpfile in the Archive
rem Staging Directory and then removes the archived document
rem content from the repository storage areas. The archive dumpfile
rem must be saved as it is the only mechanism for restoring
rem the removed content if the documents are needed
rem sometime later.
rem
rem It is suggested that the archive dumpfiles be periodically moved out
rem of the Archive Staging Directory and placed on some
rem offline storage media such as a tapes or optical devices.
rem
rem
rem This can be done by periodically backing up the Archive Staging
rem Directory to tape via a standard unix utility (tar, cpio ..)
rem or your favorite third-party backup software, and then removing
rem the backed up archive dumpfiles.
rem
rem The archive methods (post_archive, pre_restore, and post_restore
rem scripts) are called as a mechanism for automating the
rem process of moving archive dumpfiles in and out of the
rem staging area.
rem
rem NOTE THAT ONCE A DUMPFIL IS REMOVED from the Archive Staging
rem Directory, procedures must be put in place to retrieve that
rem dumpfile if a restore request is made on an archived document.
rem The restore procedure requires that the dumpfile be placed
rem back in the Archive Staging Directory in order to re-load
rem the archived content.
rem
rem When the dmarchive program is run, it requires that the archive
rem dumpfiles which contain needed content data either be already
rem accessible in the Archive Staging directory at the time
rem the restore request is seen, or that they be automatically
rem restored by the invocation of the pre_restore
rem method.
rem
rem The pre_restore method is called once for each needed archive
rem dumpfile prior to a restore operation. Upon exit the specified
rem dumpfile must be available in the Archive Staging Directory
rem for the restore load operation in order for the restore
rem request to proceed.
rem
rem A simple example of the pre_restore function would be to add the
rem shell commands at the end of this script to copy the dumpfile
rem back from the tape device to which it was written by the
rem post_archive method:
rem
rem     if [ ! -f $filename ]
rem     then
rem         echo "`date`: pre_restore: Restoring $filename" >>
rem         /tmp/archive.log
rem         tar -xf /dev/rmt0 $filename
rem     fi
rem
rem
rem
rem Default pre_restore implementation:
```

```
rem
rem This default script does not attempt to restore dumpfiles to the
rem staging directory; it merely checks their existence.
rem
rem
rem

set filename="%1"
echo ": pre_restore:Requested file:%filename%" >> \temp\archive.log
if not exist %filename% goto not_exist
    exit
not_exist:
echo ": pre_restore: Warning: %filename% does not exist." >> \temp\archive.log
fi
exit
```

The UNIX version:

```
#!/bin/sh
#$Id: pre_restore.bat,v 1.1 1994/11/13 23:27:56 kinnard Exp $
#
# pre_restore - Called prior to each restore action, its
# intention is to allow the requested archive dumpfile to be
# restored to the archive directory from any offline storage
# area it may have been moved to by the post_archive method.
#
#The pre_restore script is invoked with one argument:
#
# $1 - The name of requested archive dumpfile, including
# the filesystem path. It must be restored to its
# original filesystem directory.
#
# This script is provided as a mechanism for automatically
# restoring archive dumpfiles from their offline storage
# area so that their archived content can be retrieved by a
# restore operation.
#
# Each archive operation produces an archive dumpfile in
# the Archive Staging Directory and then removes the archived
# document content from the repository storage areas. The
# archive dumpfile must be saved as it is the only mechanism
# for restoring the removed content if the documents are
# needed sometime later.
#
# It is suggested that the archive dumpfiles be periodically
# moved out of the Archive Staging Directory and placed on
# some offline storage media such as a tapes or optical
# devices.
#
# This can be done by periodically backing up the Archive
# Staging Directory to tape via a standard unix utility (tar,
# cpio ..) or your favorite third-party backup software, and
# then removing the backed up archive dumpfiles.
#
# The archive methods (post_archive, pre_restore, and
# post_restore scripts) are called as a mechanism for
# automating the process of moving archive dumpfiles in and
```

```
# out of the staging area.

#
# NOTE THAT ONCE A DUMPFILe IS REMOVED from the Archive
# Staging Directory, procedures must be put in place to
# retrieve that dumpfile if a restore request is made on an
# archived document. The restore procedure requires that
# the dumpfile be placed back in the Archive
#
# Staging Directory in order to re-load the archived
# content. When the dmarchive program is run, it requires
# that the archive dumpfiles which contain needed content
# data either be already accessible in the Archive Staging
# directory at the time the restore request is seen, or that
# they be automatically restored by the invocation of the
# pre_restore method.
#
# The pre_restore method is called once for each needed
# archive dumpfile prior to a restore operation. Upon exit
# the specified dumpfile must be available in the Archive
# Staging Directory for the restore load operation in order
# for the restore request to proceed.
#
# A simple example of the pre_restore function would be to
# add the shell commands at the end of this script to copy
# the dumpfile back from the tape device to which it was
# written by the post_archive method:
#
#     if [ ! -f $filename ]
#     then
#         echo "`date`: pre_restore: Restoring $filename" >> /tmp/
# archive.log
#         tar -xf /dev/rmt0 $filename
#     fi
#
#####
#
# Default pre_restore implementation:
#
# This default script does not attempt to restore dumpfiles to the staging
# directory; it merely checks their existence.
#
#####
filename="$1"

echo "`date`: pre_restore: Requested file: $filename" >> /tmp/archive.log
if [ ! -f $filename ]
then
    echo "`date`: pre_restore: Warning: $filename does not exist." >>
    /tmp/archive.log
fi

exit 0
```

The post_restore script

The Windows version:

```
@echo off
rem $Id: post_restore.bat,v 2.1 1995/11/13 23:29:06 kinnard Exp $
rem
rem post_restore - Called after each restore action, its intention is
rem to allow the archive dumpfile to be removed from the archive directory
rem since its use is now complete.
rem
rem The post_restore script is invoked with one argument:
rem
rem %1 - The name of the archive dumpfile (path included)
rem
rem This script is provided as a mechanism for automatically managing
rem archive dumpfiles that are transitioned between the
rem Archive Staging Directory and offline storage during archive
rem and restore operations.
rem
rem If the post_archive and pre_restore methods are automatically used
rem copy archive dumpfiles to offline storage and back then this method
rem can be used to automatically clean up files used in restore
rem operations.
rem
rem If the method of offline storage for archive dumpfiles is such that
rem copies of the dumpfiles are created in restore operations,
rem and if the particular dumpfile on which the current restore was
rem just performed has already been copied to offline storage,
rem then this routine can safely remove that dumpfile from
rem the Archive Staging Directory as it is no longer needed.
rem
rem If your configuration does not automatically copy all archive
rem dumpfiles to offline storage, be careful that you do not modify
rem this routine to accidentally delete an archive dumpfile that
rem has not been saved yet!
rem
rem Tip: If restore requests are handled frequently, it may make sense
rem to not automatically delete them after each restore operation,
rem but to leave them around for a little while in case a second
rem restore operation is requested which needs the same dumpfile.
rem This script could be altered to only delete files which have
rem been resident for a set amount of time.
rem
rem
rem
rem Default post_restore implementation:
rem
rem
rem set filename="%1"
rem echo ": post_restore: Restored file: %filename%" >> \temp\archive.log
rem exit
```

The UNIX version:

```
#!/bin/sh
#$Id: post_restore,v 1.1 1994/11/03 00:29:49 roger Exp $
```

```
#
# post_restore - Called after each restore action, its
# intention is to allow the archive dumpfile to be removed
# from the archive directory since its use is now complete.
#
# The post_restore script is invoked with one
# argument:
#
# $1 - The name of the archive dumpfile (path included)
#
# This script is provided as a mechanism for automatically
# managing archive dumpfiles that are transitioned between
# the Archive Staging Directory and offline storage during
# archive and restore operations.
#
# If the post_archive and pre_restore methods are being
# used to automatically copy archive dumpfiles to offline
# storage and back then this method can be used to
# automatically clean up files used in restore operations.
#
# If the method of offline storage for archive dumpfiles is
# such that copies of the dumpfiles are created in restore
# operations, and if the particular dumpfile on which the
# current restore was just performed has already been
# copied to offline storage, then this routine can safely
# remove that dumpfile from the Archive Staging Directory as
# it is no longer needed.
#
# If your configuration does not automatically copy all
# archive dumpfiles to offline storage, be careful that you
# do not modify this routine to accidentally delete an
# archive dumpfile that has not been saved yet!
#
# Tip: If restore requests are handled frequently, it may
# make sense to not automatically delete them after each
# restore operation, but to leave them around for a little
# while in case a second restore operation is requested
# which needs the same dumpfile. This script could be
# altered to only delete files which have been resident for a
# set amount of time.
#
#####
#
# Default post_restore implementation:
#
#####
filename="$1"

echo "`date`: post_restore: Restored file: $filename" >> /tmp/archive.log
```

```
exit 0
```


High-Availability Support Scripts

EMC Documentum provides a set of scripts that monitor processes to determine whether a given process is running or stopped. These scripts are installed when a Content Server installation is created. The scripts can be programmatically invoked from any commercial system management and monitoring package. This appendix describes the scripts and which processes they affect.

- [Monitoring scripts, page 589](#)
- [Processes not requiring a script, page 591](#)

Monitoring scripts

Monitoring scripts are provided for Content Server, the connection broker, the Java method server, and for the Index Agent.

The scripts must be run as the Documentum Content Server installation owner. Each script returns success (the monitored process is running) as 0 or failure (the monitored process is stopped) as a non-zero value.

[Table D-1, page 589](#), lists the processes that have monitoring scripts, the script names, and their locations and command-line syntax.

Table D-1. Monitoring Scripts

Process	Script	Syntax and Location
Content Server	ContentServer-Status	A Java program in the server-impl.jar file. The command line syntax is: <pre>java ContentServerStatus -docbase_ name repository_name -user_name user_name</pre>

Process	Script	Syntax and Location
Connection broker	dmqdocbroker	<p>On UNIX, the return value is recorded in the variable \$?. On Windows, the return value is recorded in %ERRORLEVEL%.</p> <p>Located in %DM_HOME%\bin (\$DM_HOME/bin)</p> <p>The syntax is:</p> <pre>%DM_HOME%\bin\dmqdocbroker -t host_name -c ping</pre> <p>or</p> <pre>\$DM_HOME/bin/dmqdocbroker -t host_name -c ping</pre> <p><i>host_name</i> is the name of the machine hosting the connection broker.</p> <p>On UNIX, the return value is recorded in the variable \$?. On Windows, the return value is recorded in %ERRORLEVEL%.</p>
Java method server	TestConnection	<p>A Java program in the server-impl.jar file.</p> <p>On UNIX, the command line syntax is:</p> <pre>java TestConnection host port servlet</pre> <p>On Windows, the command line syntax is:</p> <pre>java TestConnection host port</pre> <p><i>host</i> is the Java Method Server (JVM) host machine; <i>port</i> is the port the JVM is using; and <i>servlet</i> is the value in dm_server_config.app_server_name that represents the JVM.</p> <p>On UNIX, the return value is recorded in the variable \$?. On Windows, the return value is recorded in %ERRORLEVEL%.</p>
Index Agent	IndexAgentCtrl	<p>A Java program in the server-impl.jar file.</p> <p>The command line syntax is:</p> <pre>java IndexAgentCtrl -docbase_name repository_name -user_name user_name -action status</pre>

repository_name is the name of a repository served by the agent and *user_name* is the user account with which to connect to the repository.

Processes not requiring a script

The following processes do not require a separate script:

- dmbasic method server
- ACS server
- Workflow agent

Content Server monitors these processes. If any of these processes stops, Content Server restarts it automatically.

Populating and Publishing the Data Dictionary

This appendix describes how to modify the data dictionary by adding or removing information, including locale files. It also describes how to publish the data dictionary. The appendix includes the following topics:

- [Populating the data dictionary, page 593](#)
- [Publishing the data dictionary information, page 607](#)

Populating the data dictionary

When you install Content Server, by default the procedure automatically populates the data dictionary with default information for one locale and sets the name of the locale in the `dd_locales` attribute of the `docbase` config object.

To add a new locale, use the population script provided by Documentum. To add to or change the locale information for installed data dictionary locales, you can use:

- The population script provided by Documentum
- The DQL `CREATE TYPE` or `ALTER TYPE` statement

Using the population script automatically records a new locale in the `dd_locales` attribute if needed.

Only some of the data dictionary information can be set using a population script. ([Summary of settable data dictionary attributes, page 596](#), describes the data dictionary attributes that you can set in a population script.) The information that you can't set using the population script must be set using the DQL statements. Additionally, you must use DQL if you want to set data dictionary information that applies only when an object is in a particular lifecycle state.

[Using DQL statements, page 606](#), discusses using DQL to populate the data dictionary.

For a description of the default data dictionary files provided with Content Server, refer to [Default files provided by EMC Documentum, page 606](#).

Data dictionary population script

The data dictionary population script, `dd_populate.ebs`, is a Docbasic script that reads an initialization file. The initialization file contains the names of one or more data files that define the data dictionary information you want to set. (Refer to [Executing the script, page 605](#), for instructions on executing the script.)

Note: Documentum provides an initialization file and data files that populate the data dictionary with the default settings for Documentum object types. For information about these, refer to [Default files provided by EMC Documentum, page 606](#).

The initialization file

You can name the initialization file any name you like, but it must have a `.ini` extension. The structure of the initialization file is:

```
[DD_FILES]
<non-NLS data dictionary data file 1>
<non-NLS data dictionary data file 2>
. . .
<non-NLS data dictionary data file n>

[NLS_FILES]
<NLS data dictionary data file 1>
<NLS data dictionary data file 2>
. . .
<NLS data dictionary data file n>
```

The non-NSL data dictionary files contain the data dictionary information that is not specific to a locale.

The NLS data dictionary files contain the information that is specific to a locale.

You can include any number of data files in the initialization file. Typically, there is one file for the non-NLS data and an NLS data file for each locale. You can reference the data files by name or full path specification. If you reference a data file by name only, the data file must be stored in the same directory as the population script or it must be in `%DM_HOME%\bin` (`$DM_HOME/bin`).

The data files

The data files are text files. You can create them with any text editor. There are two kinds of data files:

- Non-NLS
- NLS

In the non-NLS data files, the information you define is applied to all locales. In these files, you define the information that is independent of where the user is located. For example, you would set an attribute's `is_searchable` or `ignore_immutable` setting in a non-NLS-sensitive file.

In an NLS data file, all the information is assumed to apply to one locale. In these files, you identify the locale at the beginning of the file and only the `dd type info` and `dd attr info` objects for that locale are created or changed. For example, if you set the label text for a new attribute in the German locale, the system sets `label_text` in the new attribute's `dd attr info` object in the German locale. It will not set `label_text` in the attribute's `dd attr info` objects for other locales.

If you do set NLS information in a non-NLS data file, the server sets the NLS information in the session's current locale.

If you include multiple NLS files that identify the same locale, any duplicate information in them overwrites the information in the previous file. For example, if you include two NLS data files that identify Germany (de) as the locale and each sets `label_text` for attributes, the label text in the second file overwrites the label text in the first file. Unduplicated information isn't overwritten. For example, if one German file sets information for the document type and one sets information for a custom type called proposals, no information is overwritten.

Note: Future upgrades of Content Server may overwrite any changes you make to the data dictionary information about Documentum object types. To retain those changes, use a separate data file to create them. You can then re-run that data file after the upgrade to recreate the changes.

File structure

Each data file consists of sections headed by keywords that identify the object type and, if you are defining information for an attribute, the attribute. Additionally, an NLS file must identify the locale at the beginning of the file.

To specify the locale in an NLS file, place the following key phrases as the first line in the file:

- `LOCALE=locale_name CODEPAGE=codepage_name`

locale_name is defined as a string of up to five characters. When the locale is defined in a file, the server resets the current session locale to the specified locale and the information in that data file is set for the given locale.

codepage_name is the name of the code page appropriate for the specified locale. The valid code pages for the data files provided with Content Server are:

Locale	Code Page
de (German)	ISO_8859-1
en (English)	ISO_8859-1
es (Spanish)	ISO_8859-1
fr (French)	ISO_8859-1
it (Italian)	ISO_8859-1
ja (Japanese)	Shift_JIS
ko (Korean)	EUC-KR

To define information for an object type, the format is:

```
TYPE=type_name  
<data dictionary attribute settings>
```

To define information for an attribute, the format is:

```
TYPE=type_name  
ATTRIBUTE=attribute_name  
<data dictionary attribute settings>
```

If you want to define information for multiple attributes of one object type, specify the type only once and then list the attributes and the settings:

```
TYPE=type_name  
ATTRIBUTE=attribute_name  
<data dictionary attribute settings>  
{ATTRIBUTE=attribute_name  
<data dictionary attribute settings>}
```

The setting for one data dictionary attribute settings must fit on one line.

The next section, [Summary of settable data dictionary attributes, page 596](#), lists the data dictionary attributes that you can set in data files. [Examples of data file entries, page 602](#), shows some examples of how these are used in a data file.

Summary of settable data dictionary attributes

Data dictionary information is stored in the repository in internal object types. When you set data dictionary information, you are setting the attributes of these types. Some

of the information applies only to object types, some applies only to attributes, and some can apply to either or both.

Table E-1, page 597, lists and briefly describes the data dictionary attributes that you can set using a population script.

Table E-1. Settable data dictionary attributes using population script

Attribute name	Reference in script as:	Which level	NLS-specific? Yes or no	Comments
ignore_ immutable	IGNORE_ IMMUTABLE	Attribute	No	None
allowed_ search_ops default_ search_ops default_search_ arg	ALLOWED_ SEARCH_OPS DEFAULT_ SEARCH_OP DEFAULT_ SEARCH_ARG	Attribute	No	If allowed_ search_ops is set, you must set default_search_ops. Default_search_arg, if set, must be set to one of the allowed_ search_ops.
read_only	READ_ONLY	Attribute	No	None
is_hidden	IS_HIDDEN	Attribute	No	None
is_required	IS_REQUIRED	Attribute	No	None
not_null not_null_msg not_null_enf	NOT_NULL REPORT= <i>string</i> ENFORCE= <i>string</i>	Attribute	Yes	Setting not_null sets the not_null data dictionary attribute to TRUE. Including REPORT (not_null_msg) or ENFORCE (not_null_enf) is optional. If you include both, REPORT must appear first. Valid values for ENFORCE are application and disable.

Attribute name	Reference in script as:	Which level	NLS-specific? Yes or no	Comments
map_data_string map_display_ string map_description	MAPPING_TABLE VALUE= <i>integer</i> DISPLAY= <i>string</i> COMMENT= <i>string</i>	Attribute	Yes	The key phrase MAPPING_TABLE must appear before the keywords that set the values for the mapping table. COMMENT is optional. VALUE and DISPLAY must be set. Set each once for each value that you want to map. (Refer to Examples of data file entries, page 602 for an example.)
life_cycle	LIFE_CYCLE= <i>integer</i>	Object type	No	Valid values are: <i>ValueMeaning</i> 1Currently in use 2For future use 3Obsolete
label_text	LABEL_TEXT	Object type Attribute	Yes	None
help_text	HELP_TEXT	Object type Attribute	Yes	None
comment_text	COMMENT_TEXT	Object type Attribute	Yes	None
is_searchable	IS_SEARCHABLE	Object type Attribute	No	None

Attribute name	Reference in script as:	Which level	NLS-specific? Yes or no	Comments
primary_key primary_ key_msg primary_key_enf	<p>If defined at type level:</p> <p>PRIMARY_KEY=<i>key</i></p> <p>REPORT=<i>string</i></p> <p>ENFORCE=<i>string</i></p> <p>If defined at attribute level:</p> <p>PRIMARY_KEY</p> <p>REPORT=<i>string</i></p> <p>ENFORCE=<i>string</i></p>	Object type Attribute	Yes	<p>If the primary key consists of one attribute, you can define the key at either the attribute or type level. At the type level, you must name the attribute in <i>key</i>. If you define the key at the attribute level, naming the attribute is not necessary.</p> <p>If the primary key consists of multiple attributes, you must specify the key at the type level. Provide a comma-separated list of the attributes in <i>key</i>.</p> <p>Including REPORT (primary_key_msg) or ENFORCE (primary_key_enf) is optional. If you include both, REPORT must appear first.</p> <p>Valid values for ENFORCE are application and disable.</p>

Attribute name	Reference in script as:	Which level	NLS-specific? Yes or no	Comments
unique_key unique_key_msg unique_key_enf	<p>If defined at type level: UNIQUE_KEY=<i>key</i> REPORT=<i>string</i> ENFORCE=<i>string</i></p> <p>If defined at attribute level: UNIQUE_KEY REPORT=<i>string</i> ENFORCE=<i>string</i></p>	Object type Attribute	Yes	<p>If the unique key consists of one attribute, you can define the key at either the attribute or type level. At the type level, you must name the attribute in <i>key</i>. If you define the key at the attribute level, naming the attribute is not necessary.</p> <p>If the unique key consists of multiple attributes, you must specify the key at the type level. Provide a comma-separated list of the attributes in <i>key</i>.</p> <p>Including REPORT (unique_key_msg) or ENFORCE (unique_key_enf) is optional. If you include both, REPORT must appear first.</p> <p>Valid values for ENFORCE are application and disable.</p>

Attribute name	Reference in script as:	Which level	NLS-specific? Yes or no	Comments
foreign_key foreign_key_msg foreign_key_enf	At the type level: FOREIGN_KEY= <i>key</i> REFERENCES= <i>type(attr)</i> REPORT= <i>string</i> ENFORCE= <i>string</i> At the attribute level: FOREIGN_ KEY= <i>type(attr)</i> REPORT= <i>string</i> ENFORCE= <i>string</i>	Object type Attribute	Yes	Refer to Setting foreign keys , page 601 following this table for details of defining this key.

Setting foreign keys

Like other keys, you can set a foreign key at either the type level or the attribute level.

If you set the key at the type level, you must identify which attribute of the type participates in the foreign key and which attribute in another type is referenced by the foreign key. The key phrase FOREIGN_KEY defines the attribute in the object type that participates in the foreign key. The keyword REFERENCES defines the attribute which is referenced by the foreign key. For example, suppose a data file contains the following lines:

```
TYPE=personnel_action_doc
FOREIGN_KEY=user_name
REFERENCES=dm_user(user_name)
```

These lines define a foreign key for the personnel_action_doc subtype. The key says that a value in personnel_action_doc.user_name must match a value in dm_user.user_name.

To define the same foreign key at the attribute level, the data file would include the following lines:

```
TYPE=personnel_action_doc
ATTRIBUTE=user_name
FOREIGN_KEY=dm_user(user_name)
```

REPORT and ENFORCE are optional. If you include both, REPORT must appear before ENFORCE. Valid values for ENFORCE are:

- application

- disable

Examples of data file entries

Here is an excerpt from a data file that sets non-NLS data dictionary information:

```
#set attribute level information for user_name attribute
TYPE=dm_user
ATTRIBUTE=user_name
IS_REQUIRED
DEFAULT_SEARCH_OP=contains

#set attribute level information for dm_document
TYPE=dm_document
ATTRIBUTE=acl_domain
IGNORE_IMMUTABLE
ATTRIBUTE=acl_name
IGNORE_IMMUTABLE
ATTRIBUTE=title
IS_REQUIRED
DEFAULT_SEARCH_OP=contains
ATTRIBUTE=subject
DEFAULT_SEARCH_OP=contains
ATTRIBUTE=authors
IS_REQUIRED
```

This excerpt is from a data file that defines data dictionary information for the English locale.

```
#This sets the locale to English.
LOCALE = en
CODEPAGE = ISO_8859-1

# Set Attribute Information for dm_user
TYPE = dm_user
ATTRIBUTE = alias_set_id
LABEL_TEXT = User Alias Set ID

# Set Attribute Information for dm_group
TYPE = dm_group
ATTRIBUTE = alias_set_id
LABEL_TEXT = Group Alias Set ID

# Set Attribute Information for dm_process
TYPE = dm_process
ATTRIBUTE = perf_alias_set_id
LABEL_TEXT = Performer Alias Set ID

# Set Attribute Information for dm_workflow
TYPE = dm_workflow
ATTRIBUTE = r_alias_set_id
LABEL_TEXT = Performer Alias Set ID

# Set Attribute Information for dm_activity
TYPE = dm_activity
ATTRIBUTE = resolve_type
```

```

LABEL_TEXT = Alias Resolution Type
MAPPING_TABLE
VALUE = 0
DISPLAY = Default
COMMENT = Default Resolution
VALUE = 1
DISPLAY = Package-based
COMMENT = Resolution based on packages
VALUE = 2
DISPLAY = Previous Performer-based
COMMENT = Resolution based on last performer only
ATTRIBUTE = resolve_pkg_name
LABEL_TEXT = Alias Resolution Package

# Set Attribute Information for dm_sysobject
TYPE = dm_sysobject
ATTRIBUTE = a_effective_label
LABEL_TEXT = Effective Label
ATTRIBUTE = a_effective_date
LABEL_TEXT = Effective Date
ATTRIBUTE = a_expiration_date
LABEL_TEXT = Expiration Date
ATTRIBUTE = a_effective_flag
LABEL_TEXT = Effective Flag
ATTRIBUTE = a_publish_formats
LABEL_TEXT = Publish Formats
ATTRIBUTE = a_category
LABEL_TEXT = Category
ATTRIBUTE = language_code
LABEL_TEXT = Language Code
ATTRIBUTE = authors
FOREIGN_KEY = dm_user(user_name)
REPORT = The author is not found in the repository.
ENFORCE = application

# Set attribute information for dmr_containment
TYPE = dmr_containment
ATTRIBUTE = a_contain_type
LABEL_TEXT = Containment Type
ATTRIBUTE = a_contain_desc
LABEL_TEXT = Containment Description

# Set Attribute Information for dm_assembly
TYPE = dm_assembly
ATTRIBUTE = path_name
LABEL_TEXT = Path Name

# Set Attribute Information for dm_relation
TYPE = dm_relation
ATTRIBUTE = r_object_id
LABEL_TEXT = Object ID
ATTRIBUTE = relation_name
LABEL_TEXT = Relation Name
ATTRIBUTE = parent_id
LABEL_TEXT = Parent ID
ATTRIBUTE = child_id
LABEL_TEXT = Child ID
ATTRIBUTE = child_label
LABEL_TEXT = Child Label
```

```

ATTRIBUTE = permanent_link
LABEL_TEXT = Permanent Link
ATTRIBUTE = order_no
LABEL_TEXT = Order Number
ATTRIBUTE = effective_date
LABEL_TEXT = Effective Date
ATTRIBUTE = expiration_date
LABEL_TEXT = Expiration Date
ATTRIBUTE = description
LABEL_TEXT = Description

```

This final example sets some constraints and search operators for the object types and their attributes.

```

TYPE = TypeC
PRIMARY_KEY = Attr2, Attr3
  REPORT = The primary key constraint was not met.
  ENFORCE = application
UNIQUE_KEY = Attr2, Attr3
  REPORT = The unique key constraint was not met.
  ENFORCE = application
FOREIGN_KEY = Attr1
  REFERENCES = TypeA(Attr1)
  REPORT = TypeC:Attr1 has a key relationship with TypeA:Attr1
  ENFORCE = disable
IS_SEARCHABLE = True
TYPE = TypeC
ATTRIBUTE = Attr1
  ALLOWED_SEARCH_OPS = =,<,>,<=,>=,<>, NOT, CONTAINS, DOES NOT CONTAIN
  DEFAULT_SEARCH_OP = CONTAINS
  DEFAULT_SEARCH_ARG = 3
TYPE = TypeD
LIFE_CYCLE = 3
PRIMARY_KEY = Attr1
  REPORT = Attr1 is a primary key.
  ENFORCE = disable
LABEL_TEXT = label t TypeD
HELP_TEXT = help TypeD
COMMENT_TEXT = com TypeD
IS_SEARCHABLE = True
UNIQUE_KEY = Attr1
  REPORT = Attr1 is a unique key.
  ENFORCE = application
FOREIGN_KEY = Attr1
  REFERENCES = TypeC(Attr1)
  REPORT = Attr1 has a foreign key relationship.
TYPE = TypeE
ATTRIBUTE = Attr1
  IGNORE_IMMUTABLE = True
  NOT_NULL
  ENFORCE = application
  ALLOWED_SEARCH_OPS = CONTAINS, DOES NOT CONTAIN
  DEFAULT_SEARCH_OP = CONTAINS
  DEFAULT_SEARCH_ARG = 22
  READ_ONLY = True
  IS_REQUIRED = True
  IS_HIDDEN = True
  LABEL_TEXT = Attribute 1

```

```

HELP_TEXT = This attribute identifies the age of the user.
COMMENT_TEXT = You must provide a value for this attribute.
IS_SEARCHABLE = False
UNIQUE_KEY
FOREIGN_KEY = TypeB(Attr1)
REPORT = This has a foreign key relationship with TypeB:Attr1
ENFORCE = application
FOREIGN_KEY = TypeC(Attr2)
REPORT = This has a foreign key relationship with TypeC:Attr2
ENFORCE = application

```

Executing the script

The population script is named `dd_populate.ebs` and is found in `%\DOCUMENTUM%\product\version\bin ($DOCUMENTUM/product/version/bin)`.

To execute the script:

1. Change to the `%\DOCUMENTUM%\product\version\bin ($DOCUMENTUM/product/version/bin)` directory.
2. Enter the following command line at the operating system prompt:

```
dmbasic -f dd_populate.ebs -e Entry_Point -- <repository_name>
<username> <password> <ini_filename>
```

repository_name is the name of the repository.

username is the name of the user executing the script. The user must have Sysadmin or Superuser privileges.

password is the user's password.

ini_filename is the name of the initialization file that contains the data files. This can be a full path specification or only the file name. If you include just the file name, the file must be in the same directory as the population script.

Populating a Japanese locale on a Korean host or a Korean locale on a Japanese host

To populate the Japanese data dictionary locale on a Korean host or the Korean data dictionary locale on a Japanese host, connect to Content Server from a Windows computer in the desired locale and run the data dictionary population script from that computer. For example, to install the Japanese data dictionary information in a repository on a Korean host, connect to the repository from a Japanese Windows computer and run the script from the Japanese computer.

Default files provided by EMC Documentum

EMC Documentum provides the following data dictionary files with Content Server:

- `dd_populate.ebs`
`dd_populate.ebs` is the script that reads the initialization file.
- `data_dictionary.ini`
`data_dictionary.ini` is the default initialization file.
- data files

The data files contain the default data dictionary information for Documentum object types and attributes. They are located in `%DM_HOME%\bin` (`$DM_HOME/bin`). The following data files are supplied with Content Server:

- `data_dictionary_nomnls.txt`
- `data_dictionary_en.txt`
- `data_dictionary_fr.txt`
- `data_dictionary_de.txt`
- `data_dictionary_ja.txt`
- `data_dictionary_ko.txt`
- `data_dictionary_es.txt`
- `data_dictionary_it.txt`

Using DQL statements

Use the DQL `CREATE TYPE` and `ALTER TYPE` statements to set data dictionary information if:

- You cannot add the information using a population file
- The information applies only when an object is in a particular lifecycle state
- You want to add or change information for a single object type or attribute

The DQL statements allow you to publish the new or changed information immediately, as part of the statement's operations. If you choose not to publish immediately, the change is published the next time the Data Dictionary Publisher job executes. You can also use the `publish_dd` method to publish the information. (For information about publishing, refer to [Publishing the data dictionary information, page 607](#).)

For details about using these statements, refer to [Create Type, page 71](#), and [Alter Type, page 45](#), in the *Content Server DQL Reference Manual*.

Publishing the data dictionary information

Information in the data dictionary is stored in internal object types and must be published before applications or users can access it. Publishing creates the dd common info, dd type info, and dd attr info objects that applications and users can query. After you add or change information in the data dictionary, the information must be published before the changes are accessible to users or applications.

Publishing can occur automatically, through the operations of the Data Dictionary Publisher job, or on request, using the Publish_dd method or the PUBLISH keyword.

Note: Using an API method to obtain data dictionary information (refer to the instructions in [Retrieving data dictionary information, page 71](#), in *Content Server Fundamentals*) republishes the information automatically if needed. The system checks to see if the current published information matches the internally stored data dictionary information. If not, the information is republished before query results are returned.

The data dictionary publisher job

The Data Dictionary Publisher job publishes changes to the data dictionary. Each time the job runs, it publishes:

- Changes to previously published data dictionary information
- Previously unpublished data dictionary information, such as information added to a previously published locale or information for a new locale

The job uses the value of the resync_needed attribute of dd common info objects to determine which data dictionary objects need to be republished. If the attribute is set to TRUE, the job automatically publishes the data dictionary information for the object type or attribute represented by the dd common info object.

To determine what to publish for the first time, the job queries three views:

- dm_resync_dd_attr_info, which contains information for all attributes that are not published
- dm_resync_dd_type_info, which contains information for all object types that are not published
- dm_dd_policies_for_type, which contains information for all object types that have lifecycle overrides defined

The Data Dictionary Publisher job is installed with the Documentum tool suite.

The Publish_dd method

The Publish_dd method publishes the data dictionary information. You can use the method to publish the entire data dictionary or just the information for a particular object type or a particular attribute. For information about using this method, refer to [Publish_dd, page 329](#), in the *Content Server API Reference Manual*.

The PUBLISH key phrase

PUBLISH is a key phrase in the DQL CREATE TYPE and ALTER TYPE statements. If you include PUBLISH when you execute either statement, the server immediately publishes the new or changed information and any other pending changes for the particular object type or attribute.

For example, if you include PUBLISH when you create a new object type, the server immediately publishes the data dictionary for the new object type. If you include PUBLISH in an ALTER TYPE statement, the server immediately publishes the information for the altered object type or attribute and any other pending changes for that type or attribute.

If you don't include PUBLISH, the information is published the next time one of the following occurs:

- The Publish_dd method is run to update the entire dictionary or just that object type or attribute
- The Data Dictionary Publisher job runs.
- An API method is executed to obtain data dictionary information about the changed object type or attribute.

Supported and Unsupported Formats for Full-Text Indexing

The formats listed in [Table F-1, page 609](#) are supported for full-text indexing. Formats not listed in [Table F-1, page 609](#) are not supported.

Table F-1. Supported document formats

Document format	Version
Access	Versions through 2.0
Adobe Acrobat (PDF)	Versions 2.1, 3.0 – 7.0, Japanese
Adobe FrameMaker (MIF)	Version 6.0
Adobe FrameMaker graphics (FMV)	Vector/raster through 5.0
Adobe Illustrator	Versions through 7.0, 9.0
Adobe Photoshop (PSD)	Version 4.0
Ami Draw (SDW)	Ami Draw
ANSI Text	7 & 8 bit
ASCII Text	7 & 8 bit
AutoCAD Drawing	Versions 2.5 - 2.6, 9.0 - 14.0, 2000i and 2002
AutoCAD Interchange and Native Drawing formats	DXF and DWG
AutoShade Rendering (RND)	Version 2.0
Binary Group 3 Fax	All versions
Bitmap (BMP, RLE, ICO, CUR, OS/2 DIB & WARP)	All versions
CALS Raster (GP4)	Type I and Type II

Document format	Version
Computer Graphics Metafile (CGM)	ANSI, CALS NIST version 3.0
Corel Clipart format (CMX)	Versions 5 through 6
Corel Draw (CDR with TIFF header)	Versions 2.x – 9.x
Corel Draw (CDR)	Versions 3.x – 8.x
Corel/Novell Presentations	Versions through 11.0
DataEase	Version 4.x
dBASE	Versions through 5.0
dBXL	Version 1.3
DEC WPS Plus (DX)	Versions through 4.0
DEC WPS Plus (WPL)	Versions through 4.1
DisplayWrite 2 & 3 (TXT)	All versions
DisplayWrite 4 & 5	Versions through Release 2.0
EBCDIC	
Enable	Versions 3.0, 4.0 and 4.5
Enable	Versions 3.0, 4.0 and 4.5
Encapsulated PostScript (EPS)	TIFF header only
Executable (EXE, DLL)	
Executable (Windows) NT	
First Choice	through 3.0
First Choice	Versions through 3.0
First Choice	Versions through 3.0
FoxBase	Version 2.1
Framework	3.0
Framework	Version 3.0
Framework	Version 3.0
Freelance (Windows)	Versions through Millennium 9.6
Freelance for OS/2	Versions through 2.0
GEM Paint (IMG)	No specific version
Graphics Environment Mgr (GEM)	Bitmap & vector
Graphics Interchange Format (GIF)	No specific version

Document format	Version
GZIP	All versions
Hangul	Version 97
Harvard Graphics (Windows)	Windows versions
Harvard Graphics for DOS	Versions 2.x & 3.x
Hewlett Packard Graphics Language (HPGL)	Version 2
HTML	through 3.0 (some limitations)
IBM FFT	All versions
IBM Graphics Data Format (GDF)	Version 1.0
IBM Picture Interchange Format (PIF)	Version 1.0
IBM Revisable Form Text	All versions
IBM Writing Assistant	1.01
Initial Graphics Exchange Spec (IGES)	Version 5.1
JFIF (JPEG not in TIFF format)	All versions
JPEG (including EXIF)	All versions
JustSystems Ichitaro	Versions 5.0, 6.0, 8.0 – 12.0
JustWrite	Versions through 3.0
Kodak Flash Pix (FPX)	All versions
Kodak Photo CD (PCD)	Version 1.0
Legacy	Versions through 1.1
Lotus 1-2-3 (DOS & Windows)	Versions through 5.0
Lotus 1-2-3 (OS/2)	Versions through 2.0
Lotus 1-2-3 Charts (DOS & Windows)	Versions through 5.0
Lotus 1-2-3 for SmartSuite	Versions 97 – Millennium 9.6
Lotus AMI/AMI Professional	Versions through 3.1
Lotus Manuscript	Version 2.0
Lotus PIC	All versions
Lotus Snapshot	All versions
Lotus Symphony	Versions 1.0,1.1 and 2.0
Lotus Word Pro	Versions 96 through Millennium Edition 9.6, text only
LZA Self Extracting Compress	All versions

Document format	Version
LZH Compress	All versions
Macintosh PICT1 & PICT2	Bitmap only
MacPaint (PNTG)	No specific version
MacWrite II	Version 1.1
MASS11	Versions through 8.0
Micrografx Designer (DRW)	Versions through 3.1
Micrografx Designer (DSF)	Windows 95, version 6.0
Micrografx Draw (DRW)	Versions through 4.0
Microsoft Binder	Versions 7.0-97 (only on Windows)
Microsoft Excel (Mac)	Versions 3.0 – 4.0, 98, 2001
Microsoft Excel (Windows)	Versions 2.2 through 2003
Microsoft Excel Charts	Versions 2.x - 7.0
Microsoft Multiplan	Version 4.0
Microsoft Outlook Folder (PST)	Versions 97, 98, 2002, and 2002
Microsoft Outlook Message (MSG)	All versions. The body of the message is indexed and attachments to messages are indexed.
Microsoft PowerPoint (Mac)	Versions 4.0 through 2001
Microsoft PowerPoint (Windows)	Versions 3.0 through 2003
Microsoft Project	Versions 98 - 2002 (text only)
Microsoft Rich Text Format (RTF)	All versions
Microsoft Word	Versions through 6.0
Microsoft Word	Versions through 2003
Microsoft Word (Mac)	Versions 3.0 – 4.0, 98, 2001
Microsoft WordPad	All versions
Microsoft Works	Versions through 2.0
Microsoft Works	Versions through 4.0
Microsoft Works (DOS)	Versions through 2.0
Microsoft Works (DOS)	Versions through 2.0
Microsoft Works (Mac)	Versions through 2.0
Microsoft Works (Mac)	Versions through 2.0

Document format	Version
Microsoft Works (Mac)	Versions through 2.0
Microsoft Works (Windows)	Versions through 4.0
Microsoft Works (Windows)	Versions through 4.0
Microsoft Write	Versions through 3.0
MIME Text Mail	
Mosaic Twin	Version 2.5
MultiMate	Versions through 4.0
Navy DIF	All versions
Nota Bene	Version 3.0
Novell Perfect Works	Version 2.0
Novell Perfect Works	Version 2.0
Novell Perfect Works (Draw)	Version 2.0
Novell WordPerfect	Versions through 6.1
Novell WordPerfect	Versions 1.02 through 3.0
Novell/Corel WordPerfect	Versions through 11.0
Office Writer	Versions 4.0 - 6.0
OS/2 PM Metafile (MET)	Version 3.0
Paint Shop Pro 6 (PSP)	Windows only, versions 5.0 – 6.0
Paradox (DOS)	Versions through 4.0
Paradox (Windows)	Versions through 1.0
PC Paintbrush (PCX and DCX)	All versions
PC-File Letter	Versions through 5.0
PC-File+ Letter	Versions through 3.0
Personal R:BASE	Version 1.0
PFS: Professional Plan	Version 1.0
PFS: Write	Versions A, B and C
Portable Bitmap (PBM)	All versions
Portable Graymap (PGM)	No specific version
Portable Network Graphics (PNG)	Version 1.0
Portable Pixmap (PPM)	No specific version
Postscript (PS)	Level II

Document format	Version
Professional Write	Versions through 2.1
Professional Write Plus	Version 1.0
Progressive JPEG.	No specific version
Q&A	Versions through 2.0
Q&A	Version 2.0
Q&A Write	Version 3.0
Quattro Pro (DOS)	Versions through 5.0
Quattro Pro (Windows)	Versions through 11.0
R:BASE 5000	Versions through 3.1
R:BASE System V	Version 1.0
Reflex	Version 2.0
Samna Word	Versions through Samna Word IV+
SmartWare II	Version 1.02
SmartWare II	Version 1.02
SmartWare II	Version 1.02
Sprint	Versions through 1.0
Star Office/Open Office Calc	Star Office Versions 5.2, 6.x, and 7.x; Open Office version 1.1 (text only)
Star Office/Open Office Draw	Star Office 5.2, 6.x, and 7.x; Open Office version 1.1 (text only)
Star Office/Open Office Writer	Star Office Versions 5.2, 6.x, and 7.x; Open Office version 1.1 (text only)
StarOffice / Open Office Impress	StarOffice 5.2, 6.x, and 7.x; Open Office 1.1 (text only)
Sun Raster (SRS)	No specific version
SuperCalc 5	Version 4.0
Text Mail (MIME)	
TIFF	Versions through 6
TIFF CCITT Group 3 & 4	Versions through 6
Total Word	Version 1.2
Truevision TGA (TARGA)	Version 2
Unicode Text	All versions

Document format	Version
UNIX Compress	
UNIX TAR	
UUEncode	
vCard	Version 2.1
Visio	Versions 5, 2000 and 2002
Visio (preview)	Version 4
Volkswriter 3 & 4	Versions through 1.0
VP Planner 3D	Version 1.0
Wang PC (IWP)	Versions through 2.6
WBMP	No specific version
Windows Enhanced Metafile (EMF)	No specific version
Windows Metafile (WMF)	No specific version
WML	Version 5.2
WordMARC	Versions through Composer Plus
WordPerfect Graphics (WPG & WPG2)	Versions through 2.0, 7 and 10
WordStar	Versions through 7.0
WordStar	Version 1.0
WordStar 2000	Versions through 3.0
X-Windows Bitmap (XBM)	x10 compatible
X-Windows Dump (XWD)	x10 compatible
X-Windows Pixmap (XPM)	x10 compatible
XyWrite	Versions through III Plus
ZIP	PKWARE versions through 2.04g. Files inside a ZIP file are <i>not</i> indexed.

The formats listed in [Table F-2, page 615](#) are not supported for full-text indexing. Only those formats listed in [Table F-1, page 609](#) are supported.

Table F-2. Unsupported document formats

Document format	Version
Applix Graphics	4.3, 4.4
Applix Presents	4.3, 4.4

Document format	Version
Applix Spreadsheets	4.2, 4.3, 4.4
Applix Words	4.2, 4.3, 4.4
Folio Flat File	3.1
Fujitsu Oasys	7.0
Microsoft Windows Animated Cursor	No specific version
Microsoft Windows Cursor/Icon	No specific version
SGI RGB	No specific version

Appendix G

Supported Languages for Full-Text Indexing

The following languages are supported for full-text indexing:

Table G-1. Supported languages

Language	Code	Language	Code
Afrikaans	af	Italian	it
Albanian	sq	Japanese	na
Arabic	ar	Kazakh	kk
Armenian	hy	Kirghiz	ky
Azeri	az	Korean	ko
Bangla	bn	Kurdish	ku
Basque	eu	Latin	la
Bosnian	bs	Latvian	lv
Breton	br	Letzeburgesch	lb
Bulgarian	bg	Lithuanian	lt
Byelorussian	by	Macedonian	mk
Catalan	ca	Malay	ms
Chinese_simplified	zh_cn	Maltese	mt
Chinese_traditional	zh_tw	Maori	mi
Croatian	hr	Mongolian	mn
Czech	cs	Norwegian_ Bokmaal	nb

Language	Code	Language	Code
Danish	da	Norwegian_	nn
		Nynorsk	
Dutch	nl	Polish	pl
English	en	Portuguese	pt
Esperanto	eo	Rhaeto_Romance (Romansch)	rm
Estonian	et	Romanian	ro
Faeroese	fo	Russian	ru
Farsi	fa	Sami_Northern	se
Filipino (Tagalog)	tl	Serbian	sr
Finnish	fi	Slovak	sk
French	fr	Slovenian	sl
Frisian	fy	Spanish	es
Galician	gl	Swahili	sw
Georgian	ka	Swedish	sv
German	de	Tamil	ta
Greek	el	Thai	th
Greenlandic	kl	Turkish	tr
Hausa	ha	Ukrainian	uk
Hebrew	he	Urdu	ur
Hindi	hi	Uzbek	uz
Hungarian	hu	Vietnamese	vi
Icelandic	is	Welsh	cy
Indonesian	id	Yiddish	yi
Irish_Gaelic	ga	Zulu	zu

A

- a_archive attribute, 284
- a_content_attr_name
 - dm_storage_strategy setting, 230, 260
- a_content_static attribute, 256
- a_current_status attribute, 142
- a_default_retention_date attribute, 258
- a_full_text attribute, 302
- a_last_return_code attribute, 142
- a_next_invocation attribute, 148
- a_retention_attr_name attribute, 258
- a_retention_attr_required attribute, 258
- a_silent_login attribute, 371
- a_storage_params attribute, 236
- a_storage_type attribute, 242
 - dm_turbo_store value, 236
- accepting server broadcasts, 211
- access control entries
 - described, 388
- access control lists, *see* ACLs
- access permission ACL entries, 389
- access restriction ACL entries, 389
- acl_class attribute, 398
- acl_domain attribute, 399
- acl_name attribute, 399
- acl_update_threshold (server.ini key), 97
- ACLs
 - ACL object overview, 388
 - acl_domain attribute, 399
 - acl_name attribute, 399
 - acl_update_threshold (server.ini key), 97
 - adding entries, 402
 - aliases in, 398
 - audit trail entries, interpreting, 427
 - caching, 99
 - changing
 - object name, 398
 - owners, 398
 - type default, 75
 - consistency checks, 553
 - creating, 398
 - default, 401
 - default, assigning, 401
 - default, defining, 400
 - deleting
 - external, 403
 - from repository, 403
 - internal, 77, 403
 - described, 378
 - entries, described, 388
 - entry evaluation, 392
 - external, 396
 - folder default, 400
 - internal, 396
 - macl_security_disabled attribute, 396
 - modifying, 402
 - multiple entries for one user, 395
 - naming conventions, 397
 - object name, 399
 - object types, assigning to, 401
 - orphaned, 77
 - owner of, 399
 - private, 398
 - public
 - defined, 398
 - removing entries, 403
 - removing orphans, 494
 - specifying server-level default, 402
 - system
 - defined, 398
 - TCS license and evaluation, 393
 - templates, 398
 - Trusted Content Services license
 - and, 398, 402
 - type default, 400
 - user default, 400
 - using dmclean to delete, 403
- ACS servers, 86
- activating
 - administration jobs, 465

- jobs after loading, 58
- Active Directory
 - mode requirements, 363
 - user_name mapping requirement, 358
- Addsignature method
 - auditing, 408
- addDynamicGroup method, 342
- Addsignature method
 - auditing, 408
 - parameters passed by, 439
 - tracing, 440
- admingroup group, 41, 331, 342
- Administration Encryption Key, *see* AEK (Administration Encryption Key)
- administration methods
 - MIGRATE_CONTENT, 263
 - PURGE_AUDIT, 430
 - RECOVER_AUTO_TASKS, 125
- administration, server
 - API methods, 33
 - Content Server Manager, 33
 - Documentum Administrator, 32
 - DQL statements, 33
 - privileges required, 32
 - tool suite, 34
 - tools, 458
- AEK (Administration Encryption Key)
 - audit trail entries and, 410
 - backing up, 444
 - described, 443
 - determining existence, 447
 - passphrase, changing, 447
 - putting in shared memory, 446
 - sharing among products, 444
 - single-repository distributed configurations, 444
- agent exec process, 146
 - behavior modification, 152
 - database_refresh_interval (server.ini key) and, 152
 - described, 85, 460
 - disabling all jobs, 151
 - jobs in polling cycle, setting maximum, 152
 - log file location, 153
 - log files, 542
 - max_concurrent_jobs, 152
 - override_sleep_duration, 152
 - polling interval, default, 152
 - removing log files, 510
 - tracing, turning on, 153
- agent_exec_method method, 85
- agent_launcher attribute, 85, 151
- alias sets
 - creating, 72
 - deleting, 72
 - described, 72
 - modifying, 72
- aliases
 - ACL template use, 398
 - apisession, 161
 - dm_dbo, 74
 - mounted directories, 65
 - repository owner, 74
- _all_users_names (computed attribute), 343
- ALTER TYPE statement for modifying types, 75
- annotations
 - dmclean argument for, 495
 - removing orphans, 494
- annotations, removing orphans, 77
- Apache Tomcat, 128
- API (Application Program Interface)
 - api config object, 162
 - IAPI utility, 563
 - session, 161
 - session alias, 161
 - user privileges, setting, 332
- api config object
 - apiconfig key word, 220
 - DMAPI_CONFIGURATION (dmcl.ini section) and, 174
 - dmcl.ini file and, 162
 - overview, 162
 - server config object and, 162
- API events
 - audit trail entries, 414
- api.log file, 542
- apiconfig (key word), 220
- append_to_body attribute, 438
- application access control tokens
 - enabling use, 452
 - retrieval from storage, enabling, 452
 - storage location, 455
 - token files, 453
- application codes, 383
- application events, 407, 409
 - audit trails, viewing, 412
- application permission ACL entries, 390

- application restriction ACL entries, 391
- application_access_control attribute, 452
- applications
 - api config object, 160
 - API sessions, 161
 - application permits in ACLs, 390
 - application restrictions in ACLs, 391
 - Archive method, 281
 - audit trail entries, 409
 - authentication plug-ins,
 - specifying, 367
 - clients, 160
 - connection brokers and, 218
 - disk space warnings, 190
 - dmcl.ini file changes, 181
 - enabling/disabling client caching, 192
 - file extensions, 262
 - firewalls and, 212
 - is_private attribute, usage, 338
 - Mount method, 254
 - objects, controlling, 378
 - plugin authentication, using, 367
 - private and public groups, 338
 - repository sessions, 161
 - Restore method, 281
 - shutting down servers with, 118
 - URLs, using, 245
- Arabic, 295
- archive directory, 286
- archive tool
 - archive directory, choosing, 289
 - described, 284
 - generated dump record, 285
 - generated load record, 286
 - repository operator, 288
 - verbose option, 285
- Archive tool, 466
- archiving, 227
 - See also* content-addressed storage areas
 - archive directory, choosing, 289
 - archive tool, 284
 - content files, moving, 265
 - content used in many documents, 287
 - generated dump record, 285
 - generated load record, 286
 - overview, 281
 - post_archive procedure, 287
 - post_restore procedure, 287
 - pre_restore procedure, 287
 - re-archiving restored documents, 291
 - repository operator, 288
 - setting up, 289
 - strategies, 287
- arguments passed to methods, 149
- ascii attribute, 247
- ascii attribute for blob store type, 230
- asset_class attribute, 69
- attachments
 - package control and, 80
- attributes
 - adding to types, 75
 - auditing
 - initiating, 407
 - default access permissions, 73
 - deleting, 74
 - deleting from types, 76
 - dump record, 47
 - index position, 51
 - lengthening string, 76
 - load object, 47
 - mapping to LDAP, 354
 - object and ACL connection, 399
 - permissions and, 383
 - values in State of the Repository
 - Report tool, 525
- Audit Management tool, 468
 - privileges to run, 430
- Audit method, 410
 - auditing execution of, 408
- audit trail
 - defined, 408
- audit trail entries
 - lifecycle, 419
 - querying/retrieving, 413
 - removal, auditing of, 408
 - removing, 468
 - signatures, verifying, 429
 - signing, 410
- audit trails
 - common use attributes, 413
 - generic attributes, 413
 - interpreting, 413
 - removing, 430
 - viewing, 412
- audit trial entries
 - ACLs, 427
 - groups, 427
 - workflow, 420
- auditing, 406

- Addsignature method, 408
 - Addsignature method, 408
 - application events, 409
 - attribute values, 407
 - Audit Management tool, 468
 - Audit method auditing, 408
 - audit trail, 408
 - audit trail entries
 - querying/retrieving, 413
 - audit trail entries, signing, 410
 - audit trails
 - interpreting, 413
 - removing, 430
 - audit trails, viewing, 412
 - common use attributes, 413
 - compound documents, 430
 - conflicting registrations,
 - resolving, 411
 - default auditing, 408
 - distributed configurations, 430
 - dm_purgeaudit events, 408
 - dmi_registry objects and, 409
 - events, 406
 - failed logins, 409
 - IDfAuditTrailManager interface, 412
 - signatures, verifying, 429
 - Signoff method, 408
 - stopping, 412
 - system events, 409
 - Unaudit method, 412
 - Unaudit method auditing, 408
 - audittrail attrs objects
 - removing, 430
 - auth config objects, 353
 - auth_protocol attribute, 349
 - unix_domain_used setting, 353
 - authentication
 - auth_protocol attribute, 349
 - authentication plug-ins, 346
 - certutil utility, 361
 - dm_check_password file, 347
 - domain-required mode, 349
 - domains, affect of, 348
 - failed attempts, limiting, 375
 - failure auditing, 409
 - LDAP synchronization job, 362
 - Netegrity SiteMinder Policy Server, 346
 - no-domain required mode, 348
 - password checking program,
 - creating, 350
 - plug-in modules
 - described, 366
 - secure LDAP
 - setting up, 361
 - server domain, 105
 - timeouts and, 163
 - trusted logins, 371
 - unified login, 371
 - UNIX default mechanism, 347
 - UNIX users in domains, 351
 - user names, 105
 - user, options, 345
 - user_login_name attribute, 320
 - user_source attribute, 357
 - Windows default mechanism, 348
 - authentication modules
 - plug-in identifiers, 367
 - automated workflow methods
 - Tomcat, using, 140
 - automatic activities
 - execution
 - configuring, 124
 - disabling, 125
 - recovering work items on Content Server failure, 125
 - automatic operations
 - auditing failed authentication, 409
 - file clean up, 191
 - file extensions, 262
- ## B
- backslashes, in repository connection file, 154
 - base_url attribute, 249
 - basic user privileges, 321
 - bind_type attribute, 360
 - binding modes for LDAP, 360
 - blob storage areas
 - ascii attribute, 230, 247
 - content location, identifying, 236
 - described, 230
 - maximum content size, 230
 - naming rules, 247
 - setting up, 247
 - bp_schedule_*.log files, 510
 - bp_transition_*.log files, 510

broadcasts, connection brokers
 accepting, 211

C

CA store object type, 228

ca store storage areas
 content assignment policies, 237

CA store storage areas, *see*
 content-addressed storage areas

cabinets

consistency checks, 555
 Create Cabinet user privilege, 322
 creating, 44
 default ACL, 401
 defined, 43
 editing, 44
 home, 44
 permissions to modify/delete, 44
 private, 43
 public, 43
 user default, 327

cache config objects

client check interval, setting, 196
 creating, 194
 server check interval, setting, 195

cache_element_queries attribute,
 setting, 194

cache_element_types attribute,
 setting, 194

cache_queries (dmcl.ini key), 193

caches

client_cache_size (dmcl.ini key), 175
 database_referesh_interval server.ini
 key, 102
 dump process, 53
 persistent client, 192

castore_write_max_attempts (dmcl.ini
 key), 175, 230, 260

CD-ROM files, external storage and, 233

Centera host machines, clock
 requirements, 261

certdb_location attribute, 361

certificate authorities

obtaining, 361
 obtaining list of, 365

certificate database

contents, listing, 365
 location of, 361

certutil utility, 361

change_location permission, 385

change_owner permission, 385

change_permit permission, 385

change_state permission, 386

CHECK_CACHE_CONFIG

administration method, 197

checkpoint interval

described, 112

setting, 114

cleaning up file storage areas, 269

client cache

maximum size, defining, 175

client caches

flushing, 196

client caching

enabling/disabling, 192

client capability setting, 327

client configuration, 181

changing, 172, 181

changing with Restart method, 173

connection pooling, 170

dmcl.ini file, 160, 174

docbroker_search_order (dmcl.ini
 key), 209

local areas

changing directory, 188

disabling, 183, 185

enabling, 183, 187

file clean up, 191

file naming conventions, 191

monitoring disk space, 190

Purgelocal method, 191

scope of change, 185

making dmcl.ini changes visible, 181

overriding drive letter aliases, 180

overview, 160

primary connection broker

definition, 180

scope of configuration changes, 172

short date formats, 182

short date formats, defining, 182

client local areas

changing directory, 188

default directory locations, 164

disabling, 183, 185

enabling, 183, 187

file clean up, 191

file naming conventions, 191

local_diskfull_warn attribute, 190

- local_purge_on_diskfull (DMAPI_CONFIGURATION key), 190
- monitoring disk space, 190
- overview, 164
- Purgelocal method, 191
- scope of change, 185
- client_cache_size (dmcl.ini key), 175
- client_cache_write_interval (dmcl.ini key), 199
- client_caching_enabled (dmcl.ini key), 193
- client_check_interval attribute, 196
- client_pcaching_change attribute, 197
- client_pcaching_disabled attribute, 192
- client_session_timeout (server.ini key), 101
- clients
 - api sessions and, 161
 - client_session_timeout (server.ini key), 101
 - connection brokers and, 208
 - defined, 160
 - dmcl.ini file, 164
 - finding connection brokers, 219
 - Getdocbrokermmap method, 219
 - querying api config object, 220
 - repository sessions, 161
- cluster environments
 - upd_last_chg_time_from_db (server.ini key), 100
- code page requirements
 - user names, 320
- code pages
 - authentication modules, use in, 370
 - default settings, 86
 - dump and load requirements, 47
- collections
 - Kill method and, 121
- commas, in repository connection file, 154
- comments
 - IAPI format, 569
 - IDQL format, 573
- commit_read_operations (server.ini key), 97
- common area
 - removing files from, 123
- common area, see server common area, 164
- compound documents
 - auditing, 430
- computed attributes
 - _all_users_names, 343
- concurrent_sessions
 - login tickets and, 171
- concurrent_sessions, setting value in server.ini file, 101
- Config Audit privileges, 412
- Config Audit user privileges, 323
- configuration
 - changing client, 172
 - changing server configuration, 107
 - client overview, 160
 - dmcl.ini file, 160
 - DocBrokers, 214
 - objects, 29
 - overview, 28 to 29
 - scope of session changes, 172
 - session config objects, 172
- Connect method, 167
 - connection mode, specifying, 169
- connect_pooling_enabled (dmcl.ini key), 170
- connect_retry_interval (dmcl.ini file), 163
- connect_retry_limit (dmcl.ini file), 163
- connected users, viewing, 35
- connection broker
 - monitoring status, script for, 590
- connection broker locator object
 - described, 207
- connection brokers
 - adding additional, 214
 - api config attributes for, 220
 - bypassing, 166
 - clients and, 208
 - communicating with, 112
 - defining primary, 180
 - deleting server information, 220
 - failover, 166, 209
 - finding, 219
 - firewalls and, 212
 - Getdocbrokermmap method, 207
 - Getdocbasemap method, 218
 - Getservermap method, 218
 - information about, 219
 - initialization file, 210
 - invoking initialization file, 211
 - load balancing, 166, 209
 - log files, 79
 - number of, 206

- overview, 205
- projection targets, 112
- removing log files, 510
- requesting information from, 218
- restricting server access, 211
- security, 210, 216
- server shutdown notification, 120
- servers and, 85, 208
- setting checkpoint interval for servers, 114
- setting keep_entry interval for servers, 116
- shutdown security, 211
- specifying in dmcl.ini file, 165
- specifying IP address for, 210
- starting, 213
- stopping, 215
- stopping multiple, 217
- translating IP addresses, 213
- connection config objects, 163
- connection mode
 - server-side, setting, 109
- connection pooling
 - configuring session reuse limit, 170
 - connect_recycle_interval (dmcl.ini key), 170
 - dmbasic method server, 128
 - enabling, 170
 - removing free connections, 171
- connection requests
 - gethostbyaddr (server.ini key), 99
- connection strings
 - DB2, 95
 - MS SQL Server, 95
 - Oracle, 95
 - Sybase, 95
- connection strings, for ca store storage areas, 259
- connection types, 168
 - secure_connect_default (dmcl.ini key), 168
- connections
 - concurrent_sessions setting, 101
 - connection type, specifying, 168
 - removing from pool, 171
- Consistency Checker tool
 - described, 471
 - report sample, 472
- consistency checking rules
 - overriding, 198
- consistency_checker.ebs, 471
- content addressed storage areas
 - content compression, 229
- content assignment policies
 - administering, 241
 - creating, 238
 - described, 237
 - DFC cache for, 239
 - enabling/disabling, 280
 - enforcement, 238
 - repository representation, 239
 - storage algorithm, 240
 - tracing, 279
- content compression, 224
 - dump files, for, 53
- content directories
 - creating, 64
- content duplication checking
 - behavior, 225
 - filestores, in, 224
 - hash format, 226
 - Macintosh constraint, 225
 - supporting attributes, 225
- content files
 - accessing in load operations, 52
 - archiving content used in many documents, 287
 - archiving/restoring overview, 281
 - auditing movement, 266
 - automatic file extensions, 262
 - clean up in local areas, 191
 - clean up in repository, 77
 - compression in ca store storage areas, 229
 - compression in storage, 224
 - content assignment policies, 237
 - content migration policies, 264
 - Content Warning tool, 480
 - default storage algorithm, 241
 - deleting from content-addressed storage area, 268
 - digital shredding, 227
 - dos_extension attribute and, 67
 - dump files and, 52
 - dumping encrypted, 52
 - file extensions in storage, 248
 - File Report tool, 504
 - forced deletions, 257
 - hash values, 225
 - moving, 262

- naming convention in local areas, 191
- naming conventions in file stores, 244
- r_content_hash attribute, 226
- removing from content-addressed storage, 271
- removing orphan files, 269, 499
- retention of, 243
- size and full-text indexing, 295
- storage algorithm
 - primary, 241
 - renditions, 243
- storage options, 221
- storing in turbo store, 261
- use_extensions attribute, 245, 262
- content migration policies
 - described, 264
 - log file location, 265
- content objects
 - consistency checks, 556
 - defined, 236
 - i_content_id attribute and, 269
 - is_archived attribute, 284
 - is_offline attribute, 284
 - r_content_hash attribute, 225
 - removing orphans, 269, 494
 - set_file attribute, 284
 - turbo storage and, 231
- Content Replication tool
 - arguments, 478
 - described, 477
 - report sample, 479
- Content Server
 - agent exec process, 85, 460
 - audit trails
 - interpreting, 413
 - auditing
 - application events, 409
 - stopping, 412
 - auditing, default, 408
 - changing configuration, 107
 - code page, 87
 - common area, removing file from, 87
 - common area, removing files from, 123
 - communicating with connection brokers, 208
 - configuration requirements for
 - additional, 111
 - connecting to by host name, 167
 - connecting to by name, 167
 - connecting to specific server on
 - specific host, 168
 - connection brokers and, 85
 - connection strings
 - DB2, 95
 - MS SQL Server, 95
 - Oracle, 95
 - Sybase, 95
 - connection types, 168
 - data dictionary
 - populating, 593
 - database connection string, 95
 - database_refresh_interval (server.ini key), 102
 - default_acl, setting, 401
 - defining projection targets, 112, 114
 - deleting from connection broker
 - list, 220
 - described, 83
 - distinct_query_results (server.ini key), 102
 - failover, 122
 - Getservermap method, 218
 - historical session cutoff, 99
 - history_cutoff (server.ini key), 99
 - history_sessions (server.ini key), 99
 - index_store (server.ini key), 98
 - internationalization, 86
 - java.ini file, 132
 - load balancing, 122
 - max_ftacl_cache_size (server.ini key), 99
 - monitoring status, script for, 590
 - notifying connection broker of
 - shutdown, 120
 - parent and session servers, 84
 - proximity values, defining, 117
 - Reinit method, 173
 - reinitializing, 173
 - removing server logs, 510
 - removing session logs, 510
 - Restart method, 173
 - restarting, 109
 - server config object, 162
 - server.ini file, 88
 - server_startup_sleep_time key, 104
 - session config object, 162
 - sessions, shutting down, 120
 - setting checkpoint interval, 114
 - setting keep_entry interval, 116

- shutting down, 118
 - specifying listening address, 96
 - standard arguments passed to
 - jobs, 461
 - start_index_agents (server.ini key), 104
 - starting, 87
 - starting additional, 110
 - threads, 84
 - tracing options, 543
 - version number, 525
 - viewing historical sessions, 99
 - Content Server Manager, 33
 - Content Servers
 - secure connection mode, setting, 109
 - content storage areas
 - moving files, 262
 - Content Storage Services license, 237
 - content type
 - parent_id attribute, 236
 - storage_id attribute, 236
 - Content Warning tool
 - described, 480
 - percent_full argument, 481
 - report sample, 481
 - content-addressed storage areas, 227
 - a_storage_params attribute, 236
 - CA store object type, 228
 - configuration options, 228
 - connection string, setting, 259
 - content, removing, 271
 - deleting content from, 268
 - embedding or linking content, 229
 - expired objects, removing, 517
 - Linux platform, 227
 - machine clock requirements, 261
 - Macintosh computers, 228
 - retention periods, 228
 - retention policies and, 258
 - retention requirements,
 - configuring, 257
 - setting up, 256
 - write attempts, configuring, 230, 260
 - content_dupl_pref attribute, 225
 - content-file servers
 - failover limitation, 122
 - content_hash_mode attribute, 225
 - ContentServerStatus script, 590
 - copying
 - files, and disk space, 190
 - objects, folder security and, 381
 - content-addressed storage areas
 - embedded blob use, configuring, 259
 - Create Cabinet user privilege, 322
 - Create full-text events tool
 - report sample, 486
 - Create Group user privilege, 322
 - Create method, 63
 - Create Type user privilege, 322
 - CREATE...GROUP (statement), 339
 - CREATE...OBJECT (statement)
 - creating format objects, 68
 - creating jobs with, 147
 - creating location objects, 63
 - CREATE...TYPE (statement), 73
 - creating
 - blob storage areas, 247
 - default ACLs, 401
 - directories, 64
 - dump record object, 48
 - format objects, 68 to 69
 - jobs, 146
 - linked storage areas, 250
 - location objects, 62
 - method objects, 141
 - object types, 73
 - objects, folder security and, 381
 - repositories, 39
 - creating full-text events, 482
 - custom ACLs. *See* internal ACLs, 396
- ## D
- data dictionary
 - consistency checks, 558
 - Data Dictionary Publisher job, 607
 - Data Dictionary Publisher tool, 487
 - data files supplied with Content Server, 606
 - dd_populate.ebs, 594, 605 to 606
 - default files, 606
 - initialization file, 606
 - initialization file format, 594
 - populating, 593
 - data file format, 595
 - settable attributes with population file, 596
 - setting foreign keys, 601
 - populating Japanese/Korean locales, 605

- populating using DQL, 606
- PUBLISH keyword, 608
- Publish_dd method, 608
- publishing, 607
- data dictionary data files
 - described, 595
 - examples, 602
 - settable attributes, 596
 - structure of, 595
 - using multiple, 595
- Data Dictionary Publisher, 607
- Data Dictionary Publisher tool, 487
- data ticket number, 245
- data_dictionary_de.txt, 606
- data_dictionary_en.txt, 606
- data_dictionary_es.txt, 606
- data_dictionary_fr.txt, 606
- data_dictionary.ini, 606
- data_dictionary_it.txt, 606
- data_dictionary_ja.txt, 606
- data_dictionary_ko.txt, 606
- data_dictionary_nonnls.txt, 606
- data_store (server.ini key), 98
- database connection strings
 - DB2, 95
 - described, 95
 - MS SQL Server, 95
 - Oracle, 95
 - Sybase, 95
- Database Space Warning tool, 489
- database_conn server.ini key, 95
- database_name server.ini key, 96
- database_owner server.ini key, 94
- database_refresh_interval (server.ini key), 152
- date formats, short, 182
- DB2
 - database connection strings, 95
 - object type tables
 - device, defining, 92
 - uninstalling repository, affects of, 98
- dcf_edit utility, 154
- dd_locales
 - setting, 45
- dd_populate.ebs, 606
 - described, 594
 - executing, 605
- deactivating
 - jobs, 151
 - jobs during load, 58
- default ACLs
 - assigning to objects, 401
 - cabinet, 401
 - changing type default, 75
 - creating, 401
 - default_acl attribute and, 401
 - described, 400
 - folder, 400
 - specifying at server level, 401
 - type default, 400
 - user, 400
- default storage algorithm for content, 241
- default values
 - changing default group, 333
 - public and private groups, 338
 - storage area for types, 75
- default_acl attribute, 402
- default_app_permit attribute, 383
- default_folder attribute, 327
- default_storage attribute
 - format object attribute, 242
 - type info object attribute, 242
- default_storage attribute, for formats, 69
- deferred update process, 105
- delete_object permission, 386
- deleting
 - ACLs, 403
 - attributes from types, 76
 - dequeued items, 514
 - format objects, 71
 - groups, 341
 - log files, 510
 - objects, folder security and, 381
 - renditions, 520
 - users, 334
- deletions, forced, 257
- delimiters
 - signature page defaults,
 - changing, 435
- dequeued items, deleting, 79, 514
- Desktop Client
 - dmcl.ini file location, 181
- Destroy method
 - automating with version management tool, 536
 - folder security and, 381
- DFC (Documentum Foundation Classes)
 - content assignment policy enforcement, 238

- dfc.storagepolicy.validation.interval
 - property, 239
- DFTXML, 301
- dictionaries, synonym, 311
- digital shredding
 - described, 227
- digital signatures, 440
- directories
 - archive, 286
 - creating, 64
 - format objects and, 39
 - fulltext index objects and, 39
 - location objects and, 39
 - mount point objects and, 39
 - mounted drive aliases, 65
 - storage objects and, 39
 - temp, 79
 - umask dmcl.ini key, 182
 - umask server.ini key, 100, 106
- directory paths
 - archive dump file, 284
 - content in file stores, 236, 244
 - dmcl.ini file, 174
 - dump file, 56
 - State of the Repository Report
 - tool, 525
- directory servers
 - federations and, 355
 - synchronizing repositories, 356
- disabling
 - client local areas, 183, 185
 - content assignment policies, 280
 - policy engine (DFC), 280
- disk space management
 - Content Warning tool, 480
 - Database Space Warning tool, 489
 - Swap Info tool, 529
- distinct_query_results
 - server.ini key, 102
- DistOperations job, 459
- distributed content
 - dmclean tool and, 496
 - dmfilescan tool and, 501
 - full-text indexing, 293
- distributed environment and site-specific
 - tool reports, 462
- distributed environments
 - signatures, verifying, 429
- distributed repositories
 - auditing in, 430
- distributed storage areas
 - creating, 247
 - described, 231
 - load operations and, 58
 - valid component types, 231
- distributed store storage type
 - r_component attribute, 236
- distributed stores
 - components, replacing, 278
- DM_ARCHIVE event, 281
- dm_Archive job, 466
- dm_AuditMgt job, 468
- dm_audittrail_acl objects, 409
- dm_audittrail_group objects, 409
- dm_auth_config objects, 353
- dm_browse_all group, 41, 324
- dm_browse_all_dynamic group, 42, 324
- dm_check_password file, 347
- dm_ConsistencyChecker, 471
- dm_ContentReplication job, 477
- dm_ContentWarning job, 480
- dm_crypto_boot utility, 446
- dm_crypto_change_passphrase
 - utility, 447
- dm_crypto_create
 - troubleshooting AEK with, 447
- DM_CRYPTTO_FILE environment
 - variable, 443
- dm_dbo alias, 74
- dm_DBWarning job, 489
- dm_DMClean job, 494
- dm_DMFilescan job, 499
- dm_domain_conv script, 349
- DM_ENCR_PASS prefix, 372
- dm_encrypt_password utility, 373
- dm_error utility, 34
- dm_FileReport job, 504
- dm_FTCreateEvents, 309, 482
- dm_fulltext_admin group, 42
- dm_GroupRename tool, 509
- DM_HOME_CURRENT environment
 - variable, 88, 106
- dm_job_sequence objects, 144
- dm_launch_docbroker, 214
- dm_LDAPSynchronization job, 362, 491
 - described, 356
 - trace files, 365
- dm_LDAPSynchronization tool, 491
- dm_LogPurge job, 510
- dm_move_content event, 266

- dm_MoveContent
 - configurable arguments, 264
- dm_MoveContent job
 - described, 264
- dm_netegrity, 368
- DM_PLUGIN prefix, 367
- dm_QueueMgt job, 514
- dm_relation_ssa_policy objects, 239
- dm_RenditionMgt job, 520
- DM_RESTORE event, 281
- dm_retention_managers group, 42, 324
- dm_retention_users group, 42, 324
- dm_run_dependent_jobs
 - behavior, 144
- dm_run_dependent_jobs method
 - arguments, 157
 - introduced, 144
 - return values, 145
- dm_save events
 - load operations and, 57
- DM_SERVER trace facility, 227
- dm_shutdown_repository script
 - creating server_specific, 111
 - shutting down servers with, 118
- dm_ssa_policy objects, 239
- dm_start_docbase script
 - oclean argument, 87
 - starting servers with, 87
- dm_StateofDocbase job, 525
- dm_stop_docbroker utility, 216
- dm_storage_strategy, a_content_attr_
 - name value, 230, 260
- dm_store type
 - r_status attribute, 267
- dm_superuser group, 42, 324
- dm_superuser_dynamic group, 42, 324
- dm_SwapInfo job, 529
- dm_UpdateStats job, 530
- dm_UserChgHomeDb job, 533
- dm_UserRename job, 534
- dm_VersionMgt job, 536
- DMAPI_CONFIGURATION (dmcl.ini
 - section)
 - described, 174
 - local_diskfull_limit key, 190
 - local_diskfull_warn attribute, 190
 - local_purge_on_diskfull key, 190
- dmathplug.h header file, 370
- dmathplug.lib (dmathplug.a) file, 370
- dmbasic method server
 - connection pooling, 128
 - described, 128
 - enabling, 133
 - log files, removing, 510
 - worker threads, configuring, 133
- DMCL (client library), 174
 - exception_count (dmcl.ini key), 177
 - exception_count_interval (dmcl.ini
 - key), 177
- dmcl.ini file
 - cache_queries key, 193
 - castore_write_max_attempts, 175,
 - 230, 260
 - changing, 181
 - client_cache_write_interval key, 199
 - client_caching_enabled key, 193
 - comment character, 174
 - connect_pooling_enabled key, 170
 - connect_recycle_interval key, 170
 - connection broker bypass entry, 166
 - connection broker entries, 164
 - defining disk space for local areas, 190
 - described, 160, 164
 - DMAPI_CONFIGURATION
 - section, 174
 - DMCL_MASTER section, 174
 - DOCBASE_OVERRIDE_docbase
 - section, 180
 - DOCBROKER_PRIMARY
 - section, 180
 - docbroker_search_order key, 209
 - entries, 174
 - extern_store_content_static key, 177,
 - 256
 - force_coherency_check key, 198
 - force_coherency_checks key, 177
 - local_diskfull_limit key, 190
 - local_diskfull_warn attribute, 190
 - local_purge_on_diskfull
 - attribute, 190
 - locating, 181
 - making changes visible, 181
 - nfs_enabled key, 183
 - persistent_cache_write_interval
 - key, 199
 - secure_connect_default key, 168
 - specifying master file, 174
 - terminate_on_exception key, 179
 - token_storage_enabled key, 179, 452
 - token_storage_path key, 179, 452, 455

- umask key, 182
- use_local_always key, 183
- use_local_on_copy key, 183
- dmcl.ini file, locating, 181
- DMCL_MASTER (dmcl.ini section), 174
- dmclean utility
 - Apply syntax, 272
 - deleting internal ACLs, 403
 - described, 269
 - Dmclean tool, 494
 - EXECUTE syntax, 272
 - executing generated script, 273
 - external storage and, 232
 - generated script sample, 273
 - standalone syntax, 272
- dmclfull.ini file, 174
- dmdocbroker.exe, 213
- dmfilesca utility
 - Apply syntax, 276
 - described, 269, 273
 - Dmfilesca automated tool, 499
 - EXECUTE syntax, 276
 - executing generated script, 278
 - external storage and, 232
 - force_delete argument, 276
 - generated script example, 277
 - index use, 276
 - standalone syntax, 277
- dmi_audittrail_attrs
 - privileges to query, 413
- dmi_queue_item objects, removing, 79
- dmqdocbroker script, 590
- dmtkgen utility
 - arguments, 453
 - output file, 453
 - output file, naming, 455
- DO_METHOD procedures, 127
 - See also* methods
 - creating method objects, 141
 - Java method server and, 129
 - post_archive, 287
 - post_restore, 287
 - pre_restore, 287
 - tracing, 131
- do_method servlet, 128
- DocApps
 - dump operations and, 62
 - dumping limitation, 49
- docbase config object
 - dd_locales, setting, 45
 - described, 38
 - wf_package_control_enabled attribute, 80
- docbase config objects
 - macl_security_disabled attribute, 396
 - security_mode attribute, 380
- docbase configuration
 - trust_by_default attribute, 450
 - trusted_docbases attribute, 450
- Docbase sessions
 - configuration overview, 162
- docbase_id server.ini key, 94
- docbase_name server.ini key, 94
- DOCBASE_OVERRIDE_docbase dmcl.ini section, 180
- Docbasic
 - java guidelines, 135
 - tool suite implementation, 460
- docbroker locator object
 - Getdocbrokermap method, 219
- DOCBROKER_BACKUP (dmcl.ini section), 165
- DOCBROKER_CONFIGURATION section, 214
- DOCBROKER_DEBUG_BYPASS (dmcl.ini section), 166
- DOCBROKER_PRIMARY (dmcl.ini section), 165, 180
- docbroker_search_order (dmcl.ini key), 209
- DocBrokers
 - configuration, 214
- docu group, 42, 59
- documents
 - a_archive attribute, 284
 - archiving/restoring overview, 281
 - consistency checks, 556
 - defining signature page templates for, 436
 - forced deletions, 257
 - re-archiving restored documents, 291
 - restoring, 506
 - retention of, 243
 - setting up archiving, 289
- Documentum
 - creating repositories, 39
 - repository security levels, 380
 - security overview, 377
 - tool suite, 458
 - user privileges, 321

- domain controller maps, 353
 - domain-required mode
 - converting to, 349
 - described, 349
 - domains
 - auth_protocol attribute, 349
 - authenticating in, 348
 - authenticating UNIX users in, 346, 351
 - authentication, 105
 - dm_domain_conv script, 349
 - proximity values and, 118
 - dos_extension attribute, 67
 - DQL (Document Query Language)
 - auditing and, 409
 - creating method objects, 141
 - distinct_query_results (server.ini key), 102
 - format object deletion, 71
 - format object modification, 70
 - NOFTDQL hint and distinct_query_results server.ini key, 102
 - type modification procedures, 75
 - DROP TYPE (statement), 77
 - dump and load
 - table permits, affect on, 406
 - dump files
 - code pages and, 47
 - content files and, 52
 - dump object record object, 47
 - dump record objects
 - described, 47
 - dum_operaton attribute, 49
 - include_content attribute, 53
 - specifying
 - predicate attribute, 51
 - predicate2 attribute, 51
 - type attribute, 49
 - dumping
 - compressing content, 53
 - content files, 53
 - DocApps limitation, 49
 - dump_operation attribute, 49
 - external storage, 232
 - full_docbase_dump, 49
 - job objects, 58
 - objects under retention, 48
 - partial, 49
 - repositories, 46, 61
 - sample script, 55
 - selecting objects, 51
 - specifying object types, 49
 - troubleshooting, 57, 62
 - duplicate rows in queries, 102
 - duplicating repositories with dump and load, 46
 - dynamic groups, 379
 - dm_browse_all_dynamic, 42, 324
 - dm_superuser_dynamic, 42, 324
 - is_dynamic_default attribute, 338
 - membership default, setting, 342
- ## E
- electronic mail notifications
 - disabling, 103
 - electronic signatures
 - append_to_body attribute, 438
 - customizing, 431
 - max_signatures attribute, 438
 - signature creation methods, creating, 438
 - signature page templates, creating, 440
 - tracing, 440
 - electronic signoff
 - registering for notification, 442
 - embedded blobs, configuring, 259
 - embedded blobs, in content-addressed storage
 - described, 229
 - enable_workitem_mgmt (server.ini key), 98
 - enabling
 - client local areas, 183, 187
 - encrypted content files
 - dumping, 52
 - encrypted storage areas, 224
 - encryption
 - DM_ENCR_PASS prefix, 372
 - dm_encrypt_password utility, 373
 - keys, described, 443
 - passwords, discontinuing for, 373
 - resolving compromised keys, 279
 - utilities, described, 445
 - Encryptpass method, 372
 - enforce_four_digit_year key, 102
 - ENV_CONNECT_DOCBASE (iapi/idql argument), 565, 571

- ENV_CONNECT_PASSWORD (iapi/idql argument), 564, 570
 - ENV_CONNECT_USER_NAME (iapi/idql argument), 570
 - environment variables
 - DM_CRYPTO_FILE, 443
 - DM_HOME_CURRENT, 88, 106
 - error messages
 - dm_error utility, 34
 - evaluation, of ACL entries, 392
 - event notifications
 - use_group_address (server.ini key), 100
 - events
 - application events, 409
 - auditing, 406
 - defined, 406
 - dm_signoff notification, 442
 - kinds of, 407
 - notification, disabling, 103
 - system events, 409
 - user-defined, 406
 - exception handling
 - configuring, 201
 - DMCL stack trace files, 202
 - exception_count (dmcl.ini key), 177
 - exception_count_interval (dmcl.ini key), 177
 - terminate_on_exception (dmcl.ini key), 179
 - exception_count (dmcl.ini key), 177
 - setting, 201
 - exception_count_interval (dmcl.ini key), 177
 - setting, 201
 - execute_proc permission, 386
 - execution agents
 - choosing, 129
 - default, 130
 - described, 128
 - performance, 130
 - security issues, 130
 - expiration date of jobs, 149
 - expiration_date attribute, 149
 - explicit transactions
 - caching objects and, 199
 - EXPORT_TICKET_KEY administration
 - method, 449
 - exportTicketKey (DFC method), 449
 - extended characters, in LDIF files, 331
 - extended object-level permissions
 - described, 385
 - viewing, 386
 - extended restriction ACL entries, 390
 - extended user privileges, 323
 - permissions to grant/revoke, 331
 - extern file storage areas
 - described, 233
 - extern_store_content_static (dmcl.ini key), 177, 256
 - external ACLs
 - defined, 396
 - deleting, 403
 - name format, 399
 - external applications
 - dedicated servers, 38
 - jobs and, 146
 - URLs, using, 245
 - external file store storage areas
 - example of use, 253
 - external free storage areas
 - creating, 255
 - defined, 233
 - external password checking, with LDAP, 355
 - external storage
 - a_content_static attribute, 256
 - CD-ROM files, 233
 - content dump not allowed, 232
 - content load not allowed, 232
 - extern_store_content_static (dmcl.ini key), 256
 - external free stores, 233
 - external URL stores, 233
 - optimizing retrieval performance, 255
 - renditions and, 233
 - replication constraint, 232
 - external storage areas
 - described, 232
 - setting up, 253
 - types of, 232
 - external URL storage areas
 - creating, 255
 - defined, 233
- ## F
- fail-over mechanisms
 - multiple service names in server.ini file, 96

- failed_auth_attempt attribute, 336, 375
- failover
 - connection brokers, 209
 - Content Server, 122
 - content-file server limitation, 122
- federations
 - directory servers and, 355
 - ldap config objects and, 356
- file extensions
 - implementing automatic use, 262
 - in storage areas, 248
- file formats, 66
 - See also* formats
 - format objects and, 66
 - indexing non-indexable formats, 66
- file naming conventions in client local areas, 191
- File Report tool, 504
- file store keys
 - resolving compromised, 279
- file store storage areas
 - automatic file extensions, 262
 - base_url attribute, 249
 - content assignment policies, 237
 - content compression, 224
 - content duplication checking, 224
 - creating, 248
 - described, 223
 - digital shredding, 227
 - encrypted, 224
 - file store keys, resolving
 - compromised, 279
 - is_public attribute, 223
 - location_name attribute, 236
 - private access, 223
 - public access, 223
 - use_extensions attribute, 245, 262
- file systems
 - linked stores and, 234
 - Windows NT and, 249
- file systems, external storage and, 233
- file_system_path attribute
 - character constraint, 64
- filename_modifier attribute, 69
- files
 - dmclfull.ini, 174
 - dump, moving, 57
 - removing from server common area, 87, 123
 - temporary, 79
 - umask dmcl.ini key, 182
 - umask server.ini key, 100, 106
- filestore_01 storage area, 244
- filters, for LDAP, 360
- firewalls, 212
- FIXML, 301
- Flushconnectpool method, 171
- folder default ACL, 400
- folder security
 - changing, 382
 - defined, 379, 381
- folders
 - consistency checks, 555
 - creating, 44
 - defined, 43
 - editing, 44
 - permissions to modify/delete, 44
- force_coherency_check (dmcl.ini key), 198
- force_coherency_check attribute, 198
- force_coherency_checks (dmcl.ini key), 177
- foreign keys
 - setting, 601
- format objects
 - format_class attribute, 67, 294
- format objects
 - adding, 68
 - creating, 68
 - default_storage attribute, 242
 - deleting, 71
 - described, 66
 - dos_extension attribute, 67
 - modifying, 69
 - non-indexable formats, 66
 - obtaining list of, 68
 - repository configuration, 39
 - rich media formats, 69
 - topic_format_name attribute, 66
 - topic_transform attribute, 66
- format_class attribute, 67, 294
- formats
 - formats.csv file, 67
 - indexable, 295, 609
 - internal ACL names, 397
 - listing indexable, 317
 - non-indexable, 615
 - nonindexable, 308
 - obtaining list of, 68
 - rich media, 69
- formats.csv, 67

- fragmentation, table, 489
 - ft engine config objects, 302
 - ft index agent config object, 299
 - ft index agent config objects, 302
 - FTDQL
 - distinct_query_results server.ini key, 102
 - full-text indexes
 - languages, supported, 617
 - full-text indexing, 293
 - a_full_text attribute, 302
 - ACL caching, 99
 - Arabic, 295
 - content file size, 295
 - Content Server, 297 to 298
 - creating events, 309, 482
 - described, 293, 297
 - directories, 301
 - disabling content indexing, 307
 - distributed content, 293
 - dm_fulltext_admin group, 42
 - event generation, stopping during load operations, 59
 - format_class attribute, 67, 294
 - formats, 295
 - supported, 609
 - unsupported, 615
 - ft engine config objects, 302
 - ft index agent config object, 299
 - ft index agent config objects, 302
 - full-text index, 301
 - fulltext index objects, 301
 - fulltext_location attribute, 303
 - grammatical normalization, 293, 296
 - Hebrew, 295
 - ImportDictionary.py script, 312, 314
 - index agent, 297
 - index agent, starting, 305
 - index agents, 298
 - index queue, 316
 - index server, 297, 300
 - indexable content, 293
 - indexable formats, 317
 - languages, 295
 - languages, supported, 617
 - large files, 310
 - load operations and, 57
 - location objects, 302
 - nonindexable formats, 308
 - partitioning, 307
 - processing queue items, 299
 - querying indexes, 306
 - queue item processing, 299
 - reindexing, 309
 - reindexing a repository, 482
 - right-to-left languages, 295
 - server.ini file entries, 303
 - software components, 297
 - starting index agents, 104
 - storage areas, 293
 - synonym file, 312
 - SysObjects, 293
 - thesaurus searching, 311
 - timeouts, 310
 - topic_format_name attribute, 66
 - topic_transform attribute, 66
 - turning on or off, 307
 - use_estimate_search (server.ini key), 106
 - what can be indexed, 293
 - full-text queries
 - tracing, 317
 - fulltext index objects, 301
 - repository configuration, 39
 - fulltext_location attribute, 303
 - FUNCTION_EXTENT_SIZE server.ini file section, 93
 - FUNCTION_SPECIFIC_STORAGE server.ini section, 92
- ## G
- generate_event load operations
 - parameter, 59
 - Getdocbasemap method, 218
 - Getdocbrokermap method, 219
 - gethostbyaddr (server.ini key), 99
 - Getservermap method, 167, 218
 - grammatical normalization, 293, 296
 - GRANT (statement), 332
 - Group Rename tool, 509
 - group_admin attribute, 340
 - group_table_permit attribute, 405
 - groups
 - adding, 337
 - audit trail entries, interpreting, 427
 - changing default, 333
 - consistency checks, 552
 - Create Group user privilege, 322
 - deactivated users, adding, 340

- deleting, 341
- docu, 59
- dynamic, 379
- email notifications for,
 - configuring, 100
- is_dynamic_default attribute, 379
- is_private attribute, 338
- membership setting, for dynamic groups, 338
- modifying, 340
- naming constraint, 340
- naming restrictions, 338
- obtaining count of members, 343
- obtaining list of, 343
- obtaining list of members, 343
- ownership, assigning, 338
- ownership, changing, 340
- private, 338
- privileged, 323
- public, 338
- querying, 343
- renaming, 509
- required group entries in ACLs, 391
- required group set ACL entries, 392
- Saveasnew defaults, 100
- system-created, 41

H

- Hebrew, 295
- hierarchy, repository, 43
- history_cutoff (server.ini key), 99
- history_sessions (server.ini key), 99
- home cabinet, 44
- home repositories, changing, 533
- host (server.ini key), 96
- host machine
 - connecting to specific, 167
 - dm_crypto_boot, running after restart, 446
 - TCP/IP service name in server.ini file, 96
 - version number, 525
- HTTP_POST administration method
 - adding servlet for, 123

I

- i_contents_id attribute, 269
- i_crypto_key attribute, 445

- IAPI (utility)
 - commands for, 566
 - comments, entering, 569
 - executing methods, 568
 - exiting, 569
- identifiers, 367
 - See also* plug-in identifiers
- IDfAuditTrailManager interface, 412
- IDmMethod interface, 139
- IDQL (utility)
 - buffer, clearing, 573
 - commands, 572
 - idql command syntax, 569
 - queries, entering, 573
 - quiting, 574
 - starting, 569
- ignore_client_domain key, 103
- IMPORT_TICKET_KEY administration
 - method, 449
- ImportDictionary.py script, 312
- importTicketKey (DFC method), 449
- inactivating
 - administration jobs, 465
 - jobs, 151
- inboxes
 - archive and restore requests, 284
 - deleting dequeued items from repository, 514
- index agent
 - administering, 303
 - configuration file, 299
 - described, 298
 - file mode, 299, 306
 - ft index agent config object, 299
 - migration mode, 298 to 299, 305
 - modes, 298
 - modex, 305
 - monitoring status, script for, 590
 - normal mode, 298 to 299, 305
 - processing queue items, 299
 - queue item processing, 299
 - starting, 304 to 305
 - stopping, 304
- index partitioning, 307
- index positions, 51
- index queue, 316
- index server
 - described, 300
 - DFTXML, 301
 - FIXML, 301

- full-text index, 301
 - starting, 304
 - stopping, 304
 - index_store (server.ini key), 98
 - indexable formats, 295, 317
 - IndexAgentCtrl script, 590
 - indexes (type), specifying location, 98
 - indexes, for dmfilesan, 276
 - indexing large files, 310
 - information
 - connection broker, 218
 - format objects, 66
 - server startup file, 38
 - State of the Repository Report
 - tool, 525
 - tool reports, 541
 - tracing, 541
 - initialization files
 - connection broker
 - format, 210
 - invoking, 211
 - dmcl.ini, 174
 - server.ini, 88
 - installations
 - described, 28
 - making dmcl.ini changes visible, 181
 - repository security levels, 380
 - security overview, 377
 - user privileges, 321
 - internal ACLs
 - deleting, 403
 - described, 396
 - names, 397
 - object names and, 399
 - internationalization
 - Content Server, 86
 - dump and load requirements, 47
 - group names, 338
 - plug-in authentication module code
 - page, 370
 - short date formats, 182
 - short date formats, defining, 182
 - user names, 320, 325
 - IP addresses
 - for connection brokers, 210
 - specifying in server.ini, 96
 - translating, 212
 - is_archived attribute, 284
 - is_dynamic attribute, 379
 - is_dynamic_default attribute, 338, 342, 379
 - is_inactive attribute, 151
 - is_offline attribute, 284
 - is_private attribute, 338
 - is_public attribute, 223
- ## J
- Japanese locales
 - populating on Korean host, 605
 - Java method server
 - described, 128
 - method object requirements, 140
 - monitoring status, script for, 590
 - servlets, adding, 123
 - shutdown.sh script, 129
 - starting/stopping, 129
 - storage location for methods, 140
 - Tomcat, using, 140
 - java methods
 - execution agents for, 129
 - guidelines, 135
 - Java methods
 - storage location, 140
 - java.ini file, 132
 - job reports, indexing and, 463
 - job sequences
 - creating, 147
 - dcf_edit utility, 154
 - described, 143
 - dm_run_dependent_jobs method
 - execution, 157
 - execution, 144
 - failure recovery, 156
 - repository connection file, 145, 153
 - repository representation, 144
 - return codes, defining, 142
 - return values, 145
 - status report, described, 156
 - tool suite and, 465
 - jobs, 458
 - See also* tool suite
 - a_next_invocation, 148
 - activating/inactivating, 151
 - admingroup group and, 331
 - agent exec process, 85, 146, 152, 460
 - creating, 146
 - creating full-text events, 482
 - Data Dictionary Publisher, 607

- described, 143
- disabling all, 151
- dm_Archive, 466
- dm_AuditMgt, 468
- dm_ConsistencyChecker, 471
- dm_ContentReplication, 477
- dm_ContentWarning, 480
- dm_DBWarning, 489
- dm_DMClean, 494
- dm_DMFilescan, 499
- dm_FileReport, 504
- dm_FTCreateEvents, 482
- dm_GroupRename, 509
- dm_LDAPsynchronization, 362, 491
- dm_LogPurge, 510
- dm_MoveContent, 264
- dm_QueueMgt, 514
- dm_RenditionMgt, 520
- dm_StateofRepository, 525
- dm_SwapInfo, 529
- dm_UpdateStats, 530
- dm_UserChgHomeDb, 533
- dm_UserRename, 534
- dm_VersionMgt, 536
- dumping/loading, 58
- expiration_date attribute, 149
- is_inactive attribute, 151
- job reports, indexing and, 463
- maximum concurrent, setting, 152
- maximum executions, 149
- multi-server environments and, 147
- passing standard arguments, 149
- queueperson argument for tools, 461
- records migration, 265
- removing log files, 510
- reports, viewing, 463
- return codes for jobs in sequence, 142
- run_interval attribute, 149
- run_mode attribute, 149
- run_now attribute, 150
- scheduling, 148
- server time, use in scheduling, 148
- standard arguments passed, 461
- start_date attribute, 148
- target server, defining, 147
- tool suite implementation, 460
- tool suite log files, 463
- trace log files, 542
- validating cached data, 197

K

- keep_entry_interval, 116
- keepSLabel argument, 78
- keys
 - connection broker password, 211
 - server.ini file, 89
- keywords
 - PUBLISH, 608
- Kill method, 120
- Korean locales
 - populating on Japanese host, 605

L

- large files, indexing, 310
- LDAP
 - mapping attributes, 354
 - using multiple config objects, 358
- LDAP config object
 - bind_type attribute, 360
- ldap config objects
 - federations and, 356
- ldap config objects, multiple, 358
- LDAP directory server
 - activating synchronization job, 362
 - Active Directory, user_name
 - mapping, 358
 - authentication options, 355
 - Changepassword constraint, 355
 - defining set-up values, 359
 - deleting users, effect of, 336
 - dm_LDAPsynchronization tool, 491
 - implementing, 358
 - search bases/filters, 360
 - use with repository, 354
 - user/group entry precedence, 357
- LDAP directory servers
 - Active Directory, mode
 - requirements, 363
- LDAP synchronization, 362
- LDAP synchronization job
 - indexed LDAP attributes, use of, 360
- LDAP users
 - user_source attribute, 357
- LDAP, secure
 - certificate authorities, obtaining, 361
 - certutil utility, 361
- ldapcertdb_loc location object, 361
- LDIF files
 - described, 329

- extended characters and, 331
 - legacy files, external storage for, 233
 - lifecycles
 - audit trail entries, interpreting, 419
 - consistency checks, 559
 - log files, removing, 510
 - limits
 - keep entry interval, 116
 - local_diskfull_limit, 190
 - RDBMS timeout, 104
 - table fragmentation, 489
 - link_location attribute, 236
 - linked storage areas
 - advantages, 234
 - content location, identifying, 236
 - creating, 250
 - described, 233
 - linking, folder security and, 381
 - Linux platform, 227
 - load balancing, 38, 122
 - connection brokers, 209
 - load object record objects, 47
 - load operations
 - full-text indexing and, 57
 - generate_event parameter, 59
 - indexing-related events, turning off, 59
 - load record objects, 47
 - loading
 - accessing content files, 52
 - code pages and, 47
 - content not allowed into external storage, 232
 - DocApps and, 62
 - job objects, 58
 - new repositories, 59
 - preload utility, 60
 - registered tables, 59
 - repositories, 46, 61
 - troubleshooting, 62
 - local areas. *See* client local areas, 164
 - local_diskfull_limit in dmcl.ini file, 190
 - local_diskfull_warn in dmcl.ini file, 190
 - local_purge_on_diskfull attribute, 190
 - locales
 - adding, 593
 - dd_locales, setting, 45
 - locales, supported, 86
 - location objects
 - creating, 62 to 63
 - full-text indexing, 302
 - repository configuration, 39
 - security_type attribute, 64
 - State of the Repository Report tool, 525
 - location_name attribute, 236
 - log files
 - connection broker, 79
 - Content Server, 79
 - removing for repository cleanup, 79
 - server, 121
 - session, 79
 - Log Purge tool, 510
 - login failure, auditing, 408
 - login ticket key
 - exporting and importing, 449
 - resetting, 449
 - login tickets, 105
 - restricting use of global, 451
 - revoking in repository, 451
 - server cache size, setting, 171
 - validity period, defining, 450
 - validity period, setting, 171
 - login_ticket_timeout attribute, 171, 450
 - logs
 - agent exec process, 542
 - API tracing, 542
 - lifecycle, 510
 - removing files, 510
 - tool job files, 463
- ## M
- Macintosh
 - external storage and, 232
 - Windows NT servers and, 249
 - Macintosh computers, 228
 - Macintosh files
 - storage constraint, 225
 - mail_notification (server.ini key), 103
 - max_auth_attempt attribute, 336, 375
 - max_connection_per_session (dmcl.ini key), 163
 - max_ftacl_cache_size (server.ini key), 99
 - max_iterations attribute, 149
 - max_login_ticket_timeout attribute, 450
 - max_sessions_heap_size (server.ini key), 99
 - max_signatures attribute, 438
 - maximums

- age of renditions, 521
- age of versions, 538
- blob storage size, 230
- concurrent jobs, 152
- concurrent sessions, 102
- connection broker projection
 - targets, 114
- connections, 163
- dump cache size, 53
- failed authentication attempts, 375
- history sessions, 99
- job executions, 149
- number of subconnections, 163
- predicate attribute, 51
- Media Server
 - new repositories and, 41
- Media Transformation Services
 - new repositories and, 40
- memory
 - dump cache, 53
 - login tickets, 105
 - max_sessions_heap_size (server.ini key), 99
- messages
 - disk space warnings, 190
 - dump and load, 62
 - from tools, 464
 - session log file, 542
- method execution queue, 128
- method objects
 - consistency checks, 561
 - creating, 141
 - jobs passing standard arguments, 149
- method server
 - java.ini file, 132
 - method object requirements, 138
 - method_server_enabled (server.ini key), 103
 - method_server_threads (server.ini key), 103
- method_server_enabled (server.ini key), 103
- method_server_threads (server.ini key), 103
- methods
 - attribute settings for Content Server, 138
 - attribute settings for dmbasic method server, 138
 - attribute settings for Java method server execution, 140
 - defined, 127
 - executing automatically, 143
 - executing on demand, 142
 - execution agents
 - choosing, 129
 - described, 128
 - performance, 130
 - execution queue, 128
 - IAPI execution, 568
 - IDmMethod interface, 139
 - implementing, 134
 - output, recording, 137
 - removing results files, 510
 - security issues, 130
 - storing content, 128
 - tracing, 131
- Microsoft Performance Monitor tool, 33
- MIGRATE_CONTENT administration
 - method, 263
 - auditing, 266
- minimum values
 - age of audit trails, 470
 - age of dequeued items, 515
 - severity level, 546
- modifying
 - ACLs, 402
 - format objects, 69
 - groups, 340
 - server config objects, 107
 - types, 76
 - users, 333
- monitoring scripts, 589
- Mount method, 254
- mount point objects
 - alias drive letters and, 65
 - overriding drive letter aliases, 180
 - repository configuration, 39
- mount points
 - creating, 65
- moving
 - content through archive, 285
 - contents through restoring, 286
 - dump files, 57, 286
 - objects, folder security and, 381
 - repositories, 46
- MS SQL Server
 - database connection strings, 95
 - database_name server.ini key, 96

N

naming conventions
 ACLS, 397
 archive dump files, 285
 blob storage areas, 247
 data ticket values, 245
 file extensions, 248
 files in client local areas, 191
 log purge reports, 511
 storage area subdirectories, 275
 storage objects, 61
 user objects, 320
 win_preferred_alias, 65

native and secure connection mode setting, 109

native connection mode setting, 109

native connections, 168

Netegrity
 plug-in authentication module, 368

Netegrity SiteMinder Policy Server, 346

network service names, 110

nfs_enabled (dmcl.ini key), 183

nfs_enabled attribute, 183

no-domain required mode, 348

NOFTDQL hint, 102

non-indexable formats, implementing
 indexing for, 66

None user privilege, 322

nonindexable formats, 308

notifications
 disabling, 103

number (quantity of)
 concurrent sessions, 101
 connection brokers, 206
 historical sessions, 99
 login tickets, 105

numbers
 data ticket, 245
 docbase_id, 94
 port, for connection brokers, 205
 proximity values, 117

O

object replication
 external store constraint, 232

object type indexes
 consistency checks, 560

object types
 ACLs, assigning, 401

consistency checks, 558

default_storage attribute, 242

dropping from repository, 76

modifying, 74

privileges to modify/create, 73

object-level permissions
 basic permissions, 383
 described, 383
 extended permissions, 385

objects
 application control of, 383
 application-level control of, 378
 creating and folder security, 381
 deleting and folder security, 381
 dump object record, 47
 dump record, 47
 format, 39
 fulltext index, 39
 load object record, 47
 load record, 47
 location, 39
 mount point, 39
 moving and folder security, 381
 owner access evaluation, 393
 storage, 39
 superuser access to, 395

operator_name attribute
 described for archive tool, 288
 tool messages and, 461

Oracle
 database connection strings, 95
 database_name server.ini key, 96
 dm_dbo alias, 74
 object type tables
 size, defining, 93
 tablespace, defining, 92

Oracle Intranet Directory
 attributes, indexing, 360

orphan content files, 77

orphaned annotations, 77

owner_table_permit attribute, 405

P

package_control attribute, 80

packages
 component names, controlling use
 of, 80

parent server
 described, 84

- reinitializing, 186
- parent_id attribute, 236
- partitions, full-text, 307
- pass_standard_argument attribute, 149
- passphrases, changing, 447
- password checking program
 - creating, 350
 - requirements, 351
- passwords
 - change constraint with LDAP, 355
 - changing encrypted, 373
 - connection broker shutdown, 211
 - server authentication, 105
 - stopping encrypted use, 373
- performance
 - auditing and, 468
 - blob storage areas, 230
 - dmclean and dmfilescan, 269
 - dump operations, 53
 - external storage plug-in, 233
 - inboxes and, 514
 - large archive files, 287
 - load balancing, 38
 - load tracing and, 62
 - trace level 11, 547
 - turbo storage, 231
 - type indexes, 45
- performance of queries, 80
- permissions
 - base object-level, 383
 - extended object level, 385
 - multiple entries for single user, resolving, 395
- permits, registered tables, 380
- persistent client caches
 - cache config objects, creating, 194
 - cached objects backup, 198
 - client_pcaching_change attribute, 197
 - client_pcaching_disabled attribute, 192
 - consistency checking rules, overriding, 198
 - described, 192
 - troubleshooting, 199
- persistent client caching
 - client check interval, setting, 196
 - enabling/disabling, 192
 - flushing a cache, 196
 - server check interval, setting, 195
- persistent object cache
 - automatic back up, 198
- persistent_cache_write_interval (dmcl.ini key), 199
- plug-in authentication module
 - implementing custom, 368
- plug-in authentication modules
 - code page, defining, 370
 - described, 366
 - dmauthplug.h file, 370
 - dmauthplug.lib file, 370
 - identifiers, 367
 - identifying to server, 367
 - Netegrity, 368
 - trace files, 370
- plug-in identifiers
 - defining, 367
 - Netegrity, 368
- policy engine (DFC)
 - disabling, 280
- policy engine, content assignment
 - policies, 238
- port number for connection brokers, 205
- post_archive (DO_METHOD procedure), 287
- post_restore (DO_METHOD procedure), 287
- pre_restore (DO_METHOD procedure), 287
- precedence, LDAP directory server, 357
- predicate attribute, 51
- predicate2 attribute, 51
- preload utility, 60
- preserve_existing_types (server.ini key), 104
- primary content
 - storage algorithm, 241
- printing
 - from repository, 81
- private
 - ACLs
 - creating, 399
 - defined, 398
 - cabinets, 43
 - file store storage areas, 223
 - groups, 338
- private file stores, 223
- privileged groups, 323
- privileges, *see* user privileges
- process_report_admin group, 43
- programs

- shutting down servers with, 118
- projection attributes for connection
 - broker, 113
- projection targets, 112
- properties
 - changing on signature page
 - template, 434
- proximity values
 - defining, 117
 - projection_proxval attribute, 113
- Prune method
 - automating with version management
 - tool, 536
 - folder security and, 381
 - repository cleanup, 78
- public
 - cabinets, 43
 - file store areas, 223
 - groups, 338
 - storage, 183
- public ACLs
 - defined, 398
- PUBLISH keyword, 608
- Publish_dd method, 608
- publishing
 - data dictionary, 607
- Purge Audit privileges, 430
- Purge Audit user privileges, 323
- PURGE_AUDIT administration
 - method, 430
 - auditing, 408
- Purgelocal method, 191

Q

- queries
 - distinct_query_results (server.ini key), 102
 - duplicate rows in results, 102
- query performance, 80
- querying full-text indexes, 306
- queue items
 - full-text indexing, 299
 - task_state, 299
- queue items, deleting unwanted, 79
- Queue Management tool, 514
- queue_admin group, 42
- queue_manager group, 43
- queue_processor group, 43
- queueperson

- tool argument, 461
- tool errors, 541

R

- r_component attribute, 236
- r_content_hash attribute, 225 to 226
- r_gen source attribute, 410
- r_is_public attribute, acl_update_
 - threshold and, 97
- r_status attribute, 267
- RDBMS
 - connection timeout, 104
 - Database Space Warning tool, 489
 - finding fragmented tables, 489
 - locks, releasing, 97
 - login name, 320
 - setting table permits, 405
 - table permits, 404
 - updating table statistics, 530
 - version number, 525
- rdbms_connect_retry_timeout key, 104
- records migration jobs, creating, 265
- RECOVER_AUTO_TASKS administration
 - method, 125
- retention policies
 - dm_retention_managers group, 42, 324
 - dm_retention_users group, 42, 324
- registered tables
 - affects of loading operations, 59
 - setting table permits, 405
 - table permits, 404
 - user_db_name and, 320
- reindexing repositories, 309, 482
- Reinit method, 173
- rejecting server broadcasts, 211
- relation ssa policy objects, 239
- Remove Expired Retention Objects
 - tool, 517
- removeDynamicGroup method, 342
- removing
 - aborted workflows, 77
 - dequeued items, 514
 - log files, 510
 - renditions, 520
- Rendition Management tool, 520
- renditions
 - external storage and, 233
 - removing old, 78, 520

- storage algorithm, 243
- turbo storage and, 231
- replica_filestore_01 storage area, 244
- replicate_temp_store storage area, 244
- reports (tool suite), 462
- repositories
 - activating LDAP synchronization
 - job, 362
 - adding
 - format objects, 68
 - groups, 337
 - users, 324
 - AEK described, 443
 - AEK passphrase, changing, 447
 - AEK, backing up, 444
 - application access control tokens, configuring, 451
 - audit trails
 - interpreting, 413
 - removing, 430
 - auditing, 406
 - application events, 409
 - auditing, default, 408
 - changing user home, 533
 - cleaning up, 77
 - connecting, 167
 - connection config objects, 163
 - content storage options, 221
 - creating, 39
 - creating cabinets/folders, 44
 - database_owner in server.ini, 94
 - dd_locales, setting, 45
 - default_app_permit, 383
 - defined, 38
 - deleting
 - ACLs, 403
 - dequeued items, 514
 - format objects, 71
 - groups, 341
 - orphaned objects, 77
 - users, 334
 - dmbasic method server,
 - described, 128
 - docbase config object, 38
 - dumping and loading, 46, 61
 - dumping complete, 49
 - dumping partial, 49
 - enabling dmbasic method server, 133
 - encryption key, 445
 - Getdocbasemap method, 218
 - groups, system-created, 41
 - Java method server, described, 128
 - LDAP directory server, adding, 358
 - listing format objects, 68
 - loading, 57
 - loading new, 59
 - locales, adding, 593
 - login ticket keys, managing, 449
 - login tickets, configuring, 450
 - macl_security_disabled attribute, 396
 - method execution agents, 128
 - methods, implementing, 134
 - modifying
 - format objects, 69
 - groups, 340
 - users, 333
 - moving, 46
 - name in server.ini, 94
 - object defining configuration, 39
 - object types
 - modifying, 74
 - objects in new repository, 41
 - obtaining list of groups in, 343
 - organization, 43
 - query performance, 80
 - reindexing, 309
 - repository identifier in server.ini, 94
 - restoring documents, 506
 - sample dump script, 55
 - security
 - folder security setting, 382
 - server config object, 162
 - starting connection brokers, 214
 - State of the Repository tool, 525
 - storage areas, configuration
 - options, 234
 - synchronizing with directory server, 356
 - trust mode, configuring, 449
 - uninstalling, 33
 - uninstalling DB2, affects of, 98
 - user privileges
 - basic, 321
 - extended, 323
 - users
 - renaming, 334
 - users, adding multiple, 329
 - users, system-created, 41

- workflow package control,
 - configuring, 80
 - repository configuration
 - login_ticket_cutoff attribute, 451
 - repository connection file
 - creating and maintaining, 153
 - repository connection file, for job sequences, 145
 - repository encryption key, 445
 - repository name, changing, 534
 - repository operator
 - archive tool and, 288
 - tool messages and, 461
 - repository sessions
 - changing configuration, 172
 - connect_pooling_enabled (dmcl.ini key), 170
 - connect_recycle_interval (dmcl.ini key), 170
 - connection pooling, 170
 - described, 161
 - recycle limits, setting, 170
 - scope of configuration changes, 172
 - session config objects, 172
 - require_ticket attribute, 245
 - required group ACL entries, 391
 - required group set ACL entries, 392
 - RESET_TICKET_KEY administration method, 449
 - resetTicketKey (DFC method), 449
 - respository connection file
 - commas and backslashes, escaping, 154
 - dcf_edit utility, 154
 - Restart method
 - changing client configuration, 173
 - overview, 173
 - restoring content files through archive and restore, 281
 - restrict_su_ticket_login attribute, 451
 - results log files, deleting, 510
 - retainer objects
 - dump operations and, 48
 - retention periods
 - defined in ca store storage areas, 228
 - retention policies, 243
 - ca store retention periods and, 258
 - dump operations and, 48
 - retention requirements
 - configuring for ca stores, 257
 - return codes, for jobs in sequence, 142
 - REVOKE (statement), 332
 - rich media formats
 - format objects, 69
 - richmedia_enabled attribute, 69
 - root certificate authority
 - obtaining, 361
 - run_interval attribute, 149
 - run_mode attribute, 149
 - run_now attribute, 150
- ## S
- Saveasnew method, 381
 - default group assignment and, 100
 - saveasnew_retain_source_group
 - server.ini key, 100
 - schedules, changing tool, 465
 - scripts
 - archive, 287
 - ContentServerStatus, 590
 - dmclean, 269
 - dmfilescan, 274
 - dmqdocbroker, 590
 - IndexAgentCtrl, 590
 - preload utility, 60
 - sample repository dump, 55
 - TestConnection, 590
 - search bases, for LDAP, 360
 - secure connection mode
 - resetting, 109
 - secure connection mode setting, 109
 - secure connections, 168
 - secure LDAP
 - certicate database location, 361
 - setting up, 361
 - secure_connect_attribute, 109
 - secure_connect_default (dmcl.ini key), 168
 - security
 - ACL caching, 99
 - ACL naming conventions, 397
 - acl objects, 388
 - ACLs, creating, 398
 - application access control tokens, enabling use, 452
 - application control of SysObjects, 383
 - auditing, 406

- changing
 - ACL object name, 398
 - ACL owner name, 398
- connection broker shutdown, 211
- connection brokers and, 216
- Create Cabinet (user privilege), 322
- Create Type (user privilege), 322
- default ACLs, 400
- deleting ACLs, 403
- digital signatures, 440
- electronic signatures,
 - customizing, 431
- execution agents, for methods, 130
- folder, 379, 381
- granting/revoking user
 - privileges, 331
- modifying ACLs, 402
- overview, 377
- repository security levels, 380
- security argument, 88
- signing audit trail entries, 410
- Sysadmin (user privilege), 322
- table permits, 404
- user privileges, 321
- security_mode attribute, 380
- security_type attribute, 64
- SELECT (statement)
 - finding object IDs with, 403
 - finding users with, 339
- server common area
 - directory location, 164
 - overview, 164
 - removing files from, 87
 - secure writer program for, 164
- server config objects
 - changing, 107
 - connection broker projection
 - attributes, 113
 - nfs_enabled attribute, 183
 - operator_name attribute and archive
 - tool, 288
 - overview, 162
 - session config object and, 162
 - setting checkpoint_interval
 - attribute, 114
 - setting keep_entry_interval
 - attribute, 116
- server configuration
 - application access control tokens,
 - enabling, 452
 - global login tickets, restricting
 - superuser use, 451
 - login_ticket_timeout attribute, 450
 - max_login_ticket_timeout
 - attribute, 450
 - server locator objects, 167
 - server log files
 - deleting, 510
 - described, 121
 - trace messages, 542
 - server. ini file
 - start_index_agents key, 104
 - server_check_interval attribute, 195
 - server_config_name key, 104
 - server.ini
 - method_server_enabled, 103
 - server.ini file
 - acl_update_threshold, 97
 - changing, 107
 - comment character, 89
 - commit_read_operations key, 97
 - data_store key, 98
 - database_conn key, 95
 - database_name key, 96
 - database_owner key, 94
 - database_refresh_interval, 102
 - defaulted keys, 101
 - defining projection targets, 114
 - described, 88
 - docbase_id key, 94
 - docbase_name key, 94
 - enable_workitem_mgmt key, 98
 - enforce_four_digit_year key, 102
 - full-text indexing entires, 303
 - gethostbyaddr key, 99
 - history_cutoff key, 99
 - history_sessions key, 99
 - ignore_client_domain, 103
 - index_store key, 98
 - mail_notification key, 103
 - max_sessions_heap_size, 99
 - preserve_existing_types, 104
 - saveasnew_retain_source_group
 - key, 100
 - server_config_name key, 104
 - service key, 96
 - State of the Repository Report
 - tool, 525
 - upd_last_chg_time_from_db key, 100
 - use_estimate_search, 106

- use_group_address key, 100
- validate_database_user key, 101
- SERVER_STARTUP server.ini file
 - section, 89
- server_startup_sleep_time key, 104
- service (server.ini key), 96
- service name
 - connection broker, 215
 - server, 110
- services file, 214
- servlets
 - adding to Java method server, 123
- session config objects
 - described, 172
 - overview, 162
- session log files
 - deleting, 510
 - described, 542
 - dump and load messages, 62
 - tracing and, 546
- session servers, 84
- session threads, 84
- sessions
 - API, 161
 - client_session_timeout (server.ini key), 101
 - configuration, 162
 - configuring connection tries, 163
 - connect_retry_interval (dmcl.ini file), 163
 - connect_retry_limit (dmcl.ini file), 163
 - connection algorithm, 163
 - defined, 161
 - max_connection_per_session (dmcl.ini key), 163
 - max_sessions_heap_size (server.ini key), 99
 - modifying configuration, 172
 - reinitializing servers, 173
 - repository, 161
 - scope of configuration changes, 172
 - session config object, 162, 172
 - subconnections, 163
 - timeouts, 163
 - viewing historical, 99
- set_file attribute, 284
- SET_OPTIONS
 - trace_method_server flag, 131
- Setup program, 33
- severity levels, for tracing, 546
- shared libraries
 - plug-ins, 233
- shared memory, login tickets, 105
- Shift_JIS
 - NEC extensions, 87
- short date formats, 182
 - described, 182
 - UNIX defaults, 182
 - Windows default, 182
- Shutdown method, 119
- shutting down
 - connection broker, requiring
 - passwords for, 211
 - connection brokers, 215
 - servers, 118
 - sessions, 120
- signature creation method
 - custom, creating, 438
 - parameters passed to, 439
 - tracing, 440
- signature page template
 - adding/removing properties, 434
 - appearance, modifying, 435
 - append_to_body attribute, 438
 - changing delimiters, 435
 - default, described, 432
 - document types, defining for, 436
 - number of signatures,
 - configuring, 437
- signature page templates
 - creating, 440
- signature requirement support
 - simple signoffs, customizing, 441
- signature validation programs
 - customizing, 442
- signature_chk_loc attribute, 442
- signatures
 - audit trail entries, on, 410
 - digital, 440
 - verifying in distributed
 - environment, 429
- Signoff method
 - auditing, 408
- signoffs, simple
 - customizing, 441
 - signature_chk_loc attribute, 442
 - validation program, customizing, 442
- single-repository configurations
 - signatures, verifying, 429

- single-repository distributed
 - configurations
 - AEK requirements, 444
 - jobs, creating, 147
 - size
 - blob storage areas, 230
 - sleep interval (workflow agent)
 - changing, 124
 - sorting query results, 102
 - space, reclaiming
 - in repositories, 77
 - ssa policy objects, 239
 - ssl_mode attribute, 361
 - ssl_port attribute, 361
 - standard arguments for jobs, 149, 461
 - start_date attribute, 148
 - start_index_agents (server.ini key), 104
 - starting
 - additional Content Servers, 110
 - connection brokers, 213
 - Content Servers, 87, 109
 - startup.sh script, 129
 - State of the Repository tool, 525
 - statistics
 - updating, 530
 - statistics, updating, 80
 - stopping
 - connection brokers, 215
 - jobs, 151
 - servers, 118
 - storage algorithm
 - primary content, 241
 - renditions, 243
 - storage algorithms
 - DFC policy engine, 240
 - storage areas, 293
 - blob set up, 247
 - cleaning up, 269
 - configuration options, summary, 234
 - content-addressed, 227, 256
 - default, 244
 - determining state, 267
 - distributed store components,
 - replacing, 278
 - file stores, creating, 248
 - filestore_01, 244
 - full-text indexing and, 235
 - moving files, 262
 - naming conventions in file stores, 244
 - naming conventions,
 - subdirectories, 275
 - removing orphan files, 499
 - replica_filestore_01, 244
 - replicate_temp_store, 244
 - setting offline/online, 267
 - streaming_storage_01, 244
 - thumbnail_storage_01, 244
 - types, 75
 - types of, 221
 - use_extensions attribute, 248
 - storage types
 - blob store, 230
 - distributed store, 231
 - file store, 223
 - linked store, 233
 - turbo storage, 231
 - storage_id attribute, 236
 - storage objects
 - repository configuration, 39
 - streaming content files
 - storage area default, 244
 - URLs, format, 245
 - streaming_storage_01 storage area, 244
 - string attributes, lengthening, 76
 - subconnections
 - configuration, 163
 - connection config objects, 163
 - max_connection_per_session (dmcl.ini key), 163
 - subcontent objects, 231
 - success_return_codes attribute, 142
 - superuser privilege
 - access permissions for, default, 323
 - Superuser privileges
 - dm_superuser group, 42, 324
 - dm_superuser_dynamic group, 42, 324
 - Superuser user privilege
 - admingroup group and, 331
 - dumping repositories, 46
 - granting and revoking, 331
 - loading repositories, 46
 - Superuser user privileges, 322
 - superusers
 - ACL entry evaluation and, 395
 - global login tickets, restricting use of, 451
 - Swap Info tool, 529
 - Sybase

- database connection strings, 95
 - database_name server.ini key, 96
 - dm_dbo alias, 74
- synonym file, 312
- synonyms, 311
- Sysadmin user privilege, 322
- SysObjects
 - a_storage_type attribute, 242
 - access permission levels, 383
 - application-level control, 383
 - consistency checks, 554
 - determining associated ACL, 399
 - i_contents_id attribute, 269
 - indexing, 293
 - owner access evaluation, 393
 - superuser access to, 395
- system ACLs
 - creating, 399
 - defined, 398
- system administration
 - tool suite, 34
- System cabinet
 - load operation and, 59
 - location objects in, 63
 - mount point objects in, 63
 - tool reports in, 541
- system events, 406 to 407, 409
 - audit trails, viewing, 412

T

- table permits
 - affects of loading operation, 59
 - described, 380, 404
 - dump and load, effect of, 406
 - setting, 405
- tables
 - finding fragmented, 489
 - updating statistics, 530
- tablespaces
 - identifying, 98
- target server
 - defining for jobs, 147
- targets, connection broker projection, 112
- TCP/IP service name in server.ini file, 96
- Temp cabinet
 - load operations and, 59
 - tool trace logs, 541
- template ACLs, 398
 - creating, 399

- terminate_on_exception (dmcl.ini key)
 - setting, 201
 - terminate_on_exception dmcl.ini key, 179
 - TestConnection script, 590
 - thesaurus searching, 311
 - threads, session and Content Server, 84
 - thumbnail files
 - URLs, format, 245
 - Thumbnail Server
 - new repositories and, 41
 - thumbnail_storage_01 storage area, 244
 - ticket_multiplier server.ini key, 105, 171
 - tickets, number in shared memory, 105
 - timeouts, 310, 450
 - See also* validity periods
 - connection to RDBMS, 104
 - token_storage_enabled dmcl.ini key, 179, 452
 - token_storage_path dmcl.ini key, 179, 452, 455
 - Tomcat Java method server, 128
 - tool suite
 - activating/inactivating a tool, 465
 - agent exec process and, 460
 - Archive, 466
 - Audit Management, 468
 - Consistency Checker, 471
 - Content Replication, 477
 - Content Warning, 480
 - create full-text events, 482
 - Data Dictionary Publisher, 487
 - Database Space Warning, 489
 - dm_LDAPSynchronization, 491
 - Dmclean, 494
 - Dmfilescan tool, 499
 - email messages, 464
 - File Report, 504
 - Group Rename, 509
 - implementation, 460
 - inactivating a tool, 465
 - job log files, 463
 - job reports, viewing, 463
 - job sequences, using, 465
 - Log Purge, 510
 - overview, 34, 458
 - Queue Management, 514
 - queueperson argument, 461
 - Remove Expired Retention
 - Objects, 517
 - Rendition Management, 520

- reports, 462
 - running on demand, 465
 - schedule recovery, 462
 - scheduling window, 462
 - State of the Repository, 525
 - Swap Info, 529
 - tool schedules, setting, 465
 - ToolSetup (install utility), 530
 - Update Statistics tool, 530
 - UserChgHomeDb, 533
 - UserRename, 534
 - Version Management, 536
 - window_interval argument, 462
 - ToolSetup tool, 530
 - topic_format_name attribute, 66
 - topic_transform attribute, 66
 - trace files
 - DMCL exception, 202
 - LDAP synchronization jobs, 365
 - plug-in authentication modules, 370
 - Trace method, 547
 - trace_ldapsync_dfc.txt files, 365
 - trace_ldapsync_dmcl.txt files, 365
 - trace_method_server flag, 131
 - tracing
 - content duplication checking and prevention, 227
 - Content Server options, 543
 - DMCL exceptions, 202
 - full-text queries, 317
 - job execution, 153
 - method execution, 131
 - serverity levels, 546
 - workflow agent, 125
 - transactions
 - Kill method and, 121
 - loading and, 58
 - server shutdown and, 120
 - session server and, 121
 - translating IP addresses for connection brokers, 213
 - troubleshooting
 - Content Server Manager, 33
 - dump and load, 62
 - persistent client caches, 199
 - process monitoring scripts, 589
 - repository dump operation, 57
 - trust_by_default attribute, 450
 - Trusted Content Services
 - ACL evaluation and, 393
 - ACLs, creating, 398
 - ACLs, modifying, 402
 - trusted logins, 371
 - trusted logons, 349
 - trusted_docbases attribute, 450
 - turbo storage
 - described, 231
 - renditions of content, 231
 - setting up, 261
 - size constraints, 231
 - turbo storage areas
 - content location, identifying, 236
 - type default ACL, 400
 - type indexes
 - creating, 45
 - type info object type
 - default_storage attribute, 242
 - TYPE_EXTENT_SIZE server.ini file
 - section, 93
 - TYPE_SPECIFIC_STORAGE server.ini
 - section, 92
 - types, 73
 - See also* user-defined types
 - adding attributes, 75
 - changing, 75
 - Create Type user privilege, 322
 - creating, 73
 - default ACL, 400
 - default ACL, changing, 75
 - dropping attributes, 76
 - lengthening string attributes, 76
 - privileges to modify/create, 73
 - removing from repository, 76
 - specifying index location, 98
- ## U
- umask attribute, 182
 - umask dmcl.ini key, 182
 - umask server.ini key, 100, 106
 - Unaudit method, 412
 - auditing execution of, 408
 - unified login, enabling, 371
 - uninstalling a repository, 33
 - UNIX clients
 - maximum connections, 163
 - short date format, 182
 - unlinking, folder security and, 381
 - upd_last_chg_time_from_db (server.ini key), 100

- Update Statistics tool, 80, 530
- update_access_date server.ini key, 105
- updating table statistics, 530
- URLs
 - base_url attribute, 249
 - format, 245
- use_estimate_search (server.ini key), 106
- use_extensions attribute
 - appending extension, 245
 - file extensions and, 248
 - setting up storage area, 262
- use_group_address (server.ini key), 100
- use_local_always (dmcl.ini key), 183
- use_local_on_copy (dmcl.ini key), 183
- Useacl method, 401
- user authentication
 - auth_protocol attribute, 349
 - authentication name, 325
 - authentication plug-ins, 346
 - certutil utility, 361
 - dm_check_password file, 347
 - domain-required mode, 349
 - domains, affect of, 348
 - failed attempts, limiting, 375
 - LDAP synchronization job, 362
 - Netegrity SiteMinder Poliicy Server, 346
 - no-domain required mode, 348
 - options, 345
 - password checking program, creating, 350
 - plug-in modules
 - described, 366
 - secure LDAP
 - setting up, 361
 - trusted logins, 371
 - unified login, 371
 - UNIX default mechanism, 347
 - UNIX users against domain, 346
 - UNIX users in domains, 351
 - user_login_namee attribute, 320
 - user_os_domain attribute, 348
 - user_os_name attribute, 320
 - user_source attribute, 357
 - Windows default mechanism, 348
- user default ACL, 400
- user names
 - code page requirements, 320, 325
 - overview, 320
 - user_db_name attribute, 320
 - user_login_name attribute, 320
 - user_name attribute, 320
 - user_os_namee attribute, 320
- user privileges
 - audit records, viewing and, 412
 - Config Audit, 323
 - Create Cabinet, 322
 - Create Group, 322
 - Create Type, 322
 - described, 321, 380
 - GRANT statement, 332
 - None, 322
 - permissions to grant/revoke, 331
 - Purge Audit, 323
 - REVOKE statement, 332
 - setting, 326, 331
 - Superuser, 322
 - Sysadmin, 322
 - View Audit, 323
- User Renametool, 334
- user type default ACL, 400
- user_address attribute, 325
- user_auth_case key, 105
- user_auth_target key, 105
- user_db_name attribute, 320, 327
- user-defined attributes
 - adding, 75
 - dropping, 76
- user-defined types
 - adding attributes, 75
 - allowed supertypes, 73
 - changing, 75
 - changing default acl, 75
 - creating, 73
 - lengthening string attributes, 76
 - removing from repository, 76
- user_login_name attribute, 320, 325
- user_name attribute, 320, 325
- user_os_domain attribute, 348
- user_os_name attribute, 320
- user_privileges attribute, 331
- user_source attribute, 353, 357
- user_validation_location location
 - object, 347
- user_xprivileges attribute, 331
- UserChgHomeDb tool, 533
- UserRename tool, 534
- users
 - ACL entry evaluation and, 393
 - activating, 337

- adding, 324
- adding multiple, 329
- authenticating, 105
- client capability, 327
- configuring for Windows NT domain
 - authentication, 353
- consistency checks, 552
- deactivated, adding to groups, 340
- deactivating or locking, 336
- default ACL, defining, 326
- default folder, defining, 327
- deleting, 334
- email address, 325
- extended privileges, 323
- granting/revoking privileges, 331
- home repository, changing, 533
- inactivating automatically, 336
- login failure, auditing, 408
- mapping LDAP attributes, 354
- modifying, 333
- name attributes, 320
- privileges, 321, 380
- renaming, 334, 534
- system-created, 41
- user_db_name, setting, 327
- viewing connected, 35
- utilities
 - certutil, for ldap, 361
 - dm_crypto_boot, 446
 - dm_crypto_change_passphrase, 447
 - dm_crypto_create, 447
 - dm_encrypt_password, 373
 - dm_error, 34
 - dm_launch_docbroker, 214
 - dm_stop_docbroker, 216
 - dmclean, 269, 272
 - dmdocbroker.exe, 213
 - dmfilescan, 269
 - dmtkgen, 453
 - preload, 60

V

- validate_database_user server.ini key, 101
- validate_user attribute, 347
- validity periods, for login tickets
 - defining, 450
- Verifyaudit method, 429
- Verifiesignature method
 - tracing, 440

- Version Management tool, 536
- version numbers in State of the Repository
 - Report tool, 525
- versions
 - removing outdated, 78
 - Version Management tool, 536
- View Audit user privileges, 323

W

- wait_for_connect_timeout server.ini
 - key, 106
- web.xml file, trace parameter, 131
- wf_agent_worker_threads attribute, 124
 - to 125
- wf_package_control_enabled attribute, 80
- wf_sleep_interval attribute, 124
- window_interval argument, 462
- Windows clients
 - short date format, 182
- Windows domain
 - authenticating in, 348
- Windows NT
 - file systems, 249
 - Macintoshes and, 249
- work items
 - email notification, disabling, 103
 - recovering claimed work items, 125
- work queues
 - process_report_admin group, 43
 - queue_admin group, 42
 - queue_manager group, 43
 - queue_processor group, 43
- worker sessions (workflow agent)
 - changing number of, 124
- workflow agent
 - configuring, 124
 - disabling, 125
 - sleep interval, changing, 124
 - tracing, 125
 - worker sessions, changing
 - number, 124
- workflow definitions
 - package_control attribute, 80
- workflows
 - audit trail entries, interpreting, 420
 - consistency checks, 557
 - enable_workitem_mgmt (server.ini
 - key), 98
 - package name control, configuring, 80

removing aborted workflows, 77
Tomcat, using for automatic
activities, 140
use_group_address (server.ini
key), 100

workflow agent, 124
world_table_permit attribute, 405

